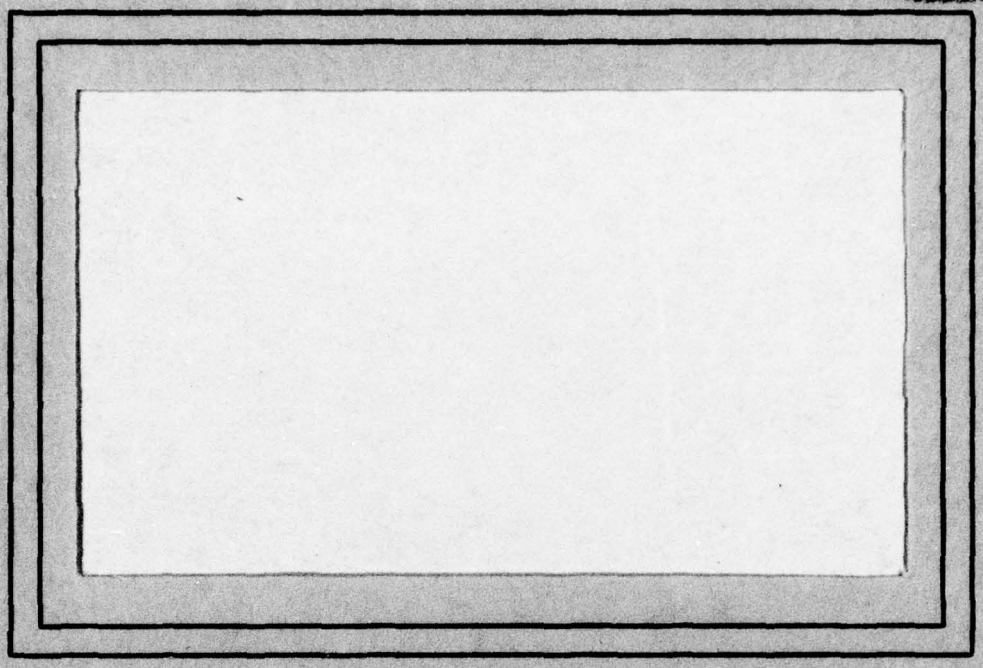
AD-A065 328 MARYLAND UNIV COLLEGE PARK COMPUTER SCIENCE CENTER F/G 5/8
MEMORY-AUGMENTED CELLULAR AUTOMATA FOR IMAGE ANALYSIS.(U)
NOV 78 C R DYER AFOSR-77-3271
UNCLASSIFIED CSC-TR-710 AFOSR-TR-79-0126 NL

1 OF 2
AD
A065328



LEVEL II



UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND
20742

LEVEL II

12

TR-710 November, 1978
AFOSR-77-3271

MEMORY-AUGMENTED CELLULAR AUTOMATA FOR IMAGE ANALYSIS

Charles R. Dyer
Computer Science Center
University of Maryland
College Park, MD 20742

ABSTRACT

This paper generalizes cellular automata by allowing the memory size associated with each cell to be a function of the input size. In particular, we define a cellular analog to the tape-bounded Turing machine for bounded cellular, pyramid cellular, and parallel/sequential automata. We focus on the case in which each cell has memory size proportional to the logarithm of the input size, showing the increased capabilities of these machines for executing a variety of basic image analysis and recognition tasks.

The support of the Directorate of Mathematical and Information Sciences, U.S. Air Force Office of Scientific Research, under Grant AFOSR-77-3271, is gratefully acknowledged, as is the help of Kathryn Riley in preparing this paper. The continued guidance and interest of Professor A. Rosenfeld is also appreciated.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

(16) (19) REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFOSR-TR-79-0126	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)*: (6) MEMORY-AUGMENTED CELLULAR AUTOMATA FOR IMAGE ANALYSIS.	5. TYPE OF REPORT & PERIOD COVERED: (9) Interim rept.,
	6. PERFORMING ORG. REPORT NUMBER: (14) CSC - TR-710
7. AUTHOR(s): (10) Charles R. Dyer	8. CONTRACT OR GRANT NUMBER(s): (15) AFOSR-77-3271
9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of Maryland, Computer Science Center, College Park, Maryland 20742	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: (16) (17) 61102F 2304 A2
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Office of Scientific Research/NM, Bolling AFB, Washington, DC 20332	12. REPORT DATE: (11) November 1978
	13. NUMBER OF PAGES: 116
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: (12) 118p.	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Cellular automata
Image processing
Pattern recognition

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This paper generalizes cellular automata by allowing the memory size associated with each cell to be a function of the input size. In particular, we define a cellular analog to the tape-bounded Turing machine for bounded cellular, pyramid cellular, and parallel/sequential automata. We focus on the case in which each cell has memory size proportional to the logarithm of the input size, showing the increased capabilities of these machines for executing a variety of basic image analysis and recognition tasks.

DD FORM 1 JAN 73 1473

403 018

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

1. Introduction

The finite-state memory requirement associated with cellular automata was originally specified in part because the automaton was defined over an infinite space and also because the objective was to design a "minimal" structure which could reproduce itself [1]. In early work investigating the language recognition capabilities of cellular automata [2-8] it was realized that an infinite cellular space is inappropriate; hence the concept of a "bounded" cellular space was introduced to force the automaton to act finitely for any fixed size input. In view of the fact that memory bounds have been extensively used as a measure of sequential automaton computations, and that there is a practical limit on the sizes of strings or arrays to be recognized, the historical precedent for a fixed amount of memory per cell is perhaps also too restrictive.

It has long been felt that the cellular array model is appropriate for efficiently performing many image analysis tasks (see [9] for an introduction to picture processing), but no systematic theoretical study of its suitability for a set of commonly used or representative operations has been conducted. In part, this is because the formal model does not approximate closely enough the properties of physical machines. For example, the finite-state memory requirement of cellular automata is too severe a restriction from a realistic point of view since

real machines have bounded size. In particular, giving each cell an amount of storage sufficient to hold the address of an arbitrary cell in the array seems reasonable. Indeed, prototype parallel image processing computers such as ILLIAC III [10] (32 by 32 with 64 bits/cell), CLIP4 [11] (16 by 12 with 32 bits/cell) and MPP [12] (128 by 128 with 256 bits/cell) all contain much more memory per cell than the logarithm of the number of processors in the array.

The effect of memory limitations on computations by sequential automata has been extensively studied, but little work has been done on the role of memory bounds in parallel computations. Most notable in this regard is the work of Stearns et al. [13] on tape-bounded Turing machine computations. Briefly, we review those results. An off-line Turing acceptor T consists of a two-way, read-only input tape and a two-way, read-write storage tape. The input string is placed between special endmarker symbols on the input tape and the read head cannot move past the endmarkers or rewrite an input symbol. A transition of T, specified deterministically by the state of the control and the symbols under each head, consists of changing state, printing a symbol on the storage tape square currently being scanned, and independently moving each head one square left, right, or not at all. We say that T is an L(n)-tape bounded Turing acceptor if for no input string of length n does T scan more than L(n) squares on the storage tape. The language accepted by T is then said to be accepted within tape

L(n). See [13-14] for a formal definition.

Stearns et al. showed that the regular sets are recognizable with L(n)=1, and furthermore, to recognize a nonregular set requires at least L(n) tape, where $\sup_{n\to\infty} \frac{L(n)}{\log \log n} > 0$ Also, L(n)=logn is necessary and $L(N)=\log^2 n$ is sufficient for recognizing the context-free languages. The context-sensitive languages are recognizable with L(n)=n.

In view of the unclosed gap in the memory requirement for recognition of the context-free languages, many researchers have focused attention on which subclasses can be recognized with L(n)=log n [15-18]. Furthermore, the equivalence of log-tape bounded Turing machines with multihead automata [19] and marker, or pebble, atuomata [17] is noteworthy as providing further evidence of the importance of this class of memory-limited computations. In addition, the implications of log-tape bounded Turing machine computations on other well-known problems have been investigated [19-21]. Analogously, theoretical results on memory-bounded cellular automata are potentially important because they yield insights into how the difficulty of a computation is related to the machine's memory capacity.

This paper studies memory-augmented cellular automata, focusing in particular on the case where each cell has memory size proportional to the logarithm of the input size. In Section 2 we define memory-augmented bounded cellular acceptors and establish the relationship between them and tape-bounded Turing

acceptors. Sections 3-5 investigate the capabilities of log-memory bounded cellular automata for performing a variety of basic one- and two-dimensional image analysis tasks. Unlike ordinary BCA's, each cell now has sufficient memory to store its own coordinates and to compute various arithmetic functions whose range is limited by the array size. Section 6 considers memory-augmented pyramid cellular automata and Section 7 discusses memory-augmented parallel/sequential automata.

2. Memory-augmented bounded cellular automata

2.1 Definitions

This section generalizes the standard definition of a bounded cellular automaton, which specifies that each cell has a finite state set, to allow the memory size associated with each cell to be a function of the input size. That is, we define a cellular analog to the tape-bounded Turing machine.

A memory-augmented cell is a Turing machine without input tape, i.e., a 5-tuple $C=(Q,\Gamma,\delta,Q_I,b)$, where Q is the finite, nonempty state set, Γ is the finite, nonempty storage tape alphabet, $\delta:(Q\times\Gamma)^3\to 2^{(Q\times\Gamma\times\{-1,0,1\})}$ is the transition function ($\delta:(Q\times\Gamma)^3\to Q\times\Gamma\times\{-1,0,1\}$ if C is deterministic), $Q_I\subseteq Q$ is the set of input states, and $b\in\Gamma$ is the blank storage tape symbol initially written on every square of the storage tape.

A memory-augmented bounded cellular automaton is a pair $M=(C,\#)$, where $C=(Q,\Gamma,\delta,Q_I,b)$ is a memory-augmented cell, one copy of which is assigned to each integer point on the real line; the copy at coordinate i is called cell i. $\#\in Q_I$ is a special boundary state. The transition function for cell i maps the current states of cells i-1, i, and i+1, and their corresponding storage tape symbols currently being scanned, into a set of triples of possible new states of cell i's finite control, new symbols written at the storage tape square where i's head is currently positioned, and directions of movement of

cell i's head. A step of computation consists of the simultaneous application of the transition function at each cell. A configuration of M is a mapping from the integers into (Q, Γ^+, j) triples specifying the current state , tape contents and head position of each cell in M. For convenience, only those tape positions which the head has visited will be included in that tape's description since the remainder of the tape is known to be blank. The configuration prior to the first time step is called the initial configuration.

The boundary state # is used in the usual way to restrict a computation to a bounded number of contiguous cells. That is, an initial configuration of M is of the form $(\#,b,1)^\infty (a_1,b,1)(a_2,b,1)\cdots(a_n,b,1)(\#,b,1)^\infty$, where $a_1a_2\cdots a_n$ is a finite, non-null string in $(Q_I-\{\#\})^+$, called the input string. Boundedness is now enforced by restricting the transition function δ to be both #-preserving and write-inhibited on storage tapes of cells in the boundary state. That is, $\delta(p,x,q,y,r,z) = (\#,w,d)$ imples $q=\#$, $w=y$, and $d=0$, and $\delta(p,x,\#,y,r,z) = (\#,y,0)$ for all p,r in Q and x,y,z in Γ. Because of these conditions we will assume without loss of generality that the string of cells is finite, and initially has the form $(\#,b,1)(a_1,b,1)\cdots(a_n,b,1)(\#,b,1)$.

M is called an L(n)-space bounded cellular automaton if L(n) is an upper bound on the number of storage tape squares visited by any cell in M given any input string of length n

and any valid sequence of steps of M. In particular, if L(n) =log n, then M is called a log-space bounded cellular automaton.

An L(n)-space bounded cellular acceptor (L(n)-space BCA) is a pair $Z=(M,Q_A)$, where M is an L(n)-space bounded cellular automaton and $Q_A \subseteq Q$ is the set of accepting states. Z's left-most non-# cell is called the accept cell or cell 1. An input string $a_1a_2...a_n$ is said to be accepted by Z if given the initial configuration $(\#,b,1), (a_1,b,1),...,(a_n,b,1),(\#,b,1)$, Z's accept cell eventually enters an accepting state after some number of time steps. The set of strings accepted by Z defines its language.

In two dimensions, an L(n)-space bounded cellular automaton is defined by a straightforward extension of the one-dimensional definition. A memory-augmented cell is defined in the same way, except the transition function is now a function of a 5-tuple of (state, storage symbol) pairs. A copy of the cell is assigned to each point in I^2, where I is the set of integers. Cell (i,j)'s next state depends on the local configurations, i.e., (state, symbol) pairs, of itself and its four nearest neighbors, cells (i-1,j), (i+1,j), (i,j-1), and (i,j+1). An ℓ-by-m input array, $n=\ell m$, defines the initial states of a rectangular block of cells which are surrounded by a border of #-cells. The accept cell in an L(n)-space bounded cellular acceptor is the upper-left corner non-# cell.

A language L is said to be accepted by a (one- or two-dimensional) L(n)-space BCA Z in $O(f(\ell,m))$ time if there exists a constant k such that every ℓ-by-m array, $n=\ell m$, in L is accepted by Z within $k \cdot f(\ell,m)$ time steps. In particular, if $f(\ell,m)=\ell+m$, then we say Z accepts L in O(diameter) time. If $f(\ell,m)=\ell m$, then we say Z accepts L in O(area) time. In the one-dimensional case $\ell=1$, so diameter equals area. Although it is usual in this case to denote $f(1,m)=f(n)=n$ as O(linear) time, we use O(diameter) in order to be consistent with the two-dimensional case.

While this definition specifies the desired formal extension of bounded cellular acceptors to include augmented memory computations, it restricts storage access to only a single square of the storage tape at each time step. In order to simplify algorithm description and emphasize arithmetic rather than logical operations as primitive, we would like the transition function to "depend" on the entire contents of a cell's storage tape. We will limit this dependence by invoking a unit time cost for the most part only for elementary operations that can be executed by an L(n)-space BCA in time proportional to L(n). (Multiplication will be the major exception.)

By analogy with the tape-reduction theorem for tape-bounded Turing machines [13], it is easily seen that a constant factor in the length of the storage tape does not affect the language acceptability of L(n)-space BCA's. Therefore, we will consider

the storage tape to be divided into a finite number of tracks, called registers, each of length L(n). The unit time criterion will be used for executing instructions such as: set the contents of register i to zero, increment the contents of register j, or copy the contents of register i into register j.

2.2 Relation to tape-bounded Turing machines

In order to compare the languages accepted by tape-bounded Turing acceptors with the languages accepted by L(n)-space BCA's, we need the concept of measurability. A function L(n) is said to be measurable if there is some off-line Turing machine T such that, given any input of length n, T will halt after a computation in which the storage tape head scans exactly L(n) squares. T is then said to construct L(n).

Theorem 2.1. The class of L(n)-space BCA languages is equivalent to the class of (n·L(n))-tape bounded Turing acceptor languages.

Proof: We give the proof for the case where the BCA is a string of cells; the generalization to two dimensions, where n is the array area, is straightforward. An L(n)-space BCA M can simulate an (n·L(n))-tape bounded Turing acceptor T as follows. First, consider the case where L(n) is measurable. Let T' be an off-line Turing machine which constructs L(n). With input x of length n, M first simulates T' by passing a marker from cell to cell to keep track of T''s input head position, while using the leftmost non-# cell's storage tape to record T''s storage tape contents and head position. When T' halts, M copies the contents of the leftmost cell's storage tape into every other cell's storage tape in M. This allows M to mark off exactly L(n) squares on the storage tape in each cell.

The simulation is now straightforward. M passes a marker from finite control to finite control reflecting the movement of T's read head. Since M's storage tapes have endmarkers L(n) squares apart and are ordered by the cells that they occur in, M can treat this length n sequence of L(n)-bounded tapes as a single storage tape of length $n \cdot L(n)$--just enough space to simulate T.

To simlulate one transition of T, the two cells marking the positions of the read and storage heads initiate signals containing the state and storage symbol located at those positions. After no more than n/2 time steps, the cell half way between these cells receives all the information it needs to determine T's transition. This cell then returns (new state, direction of read head movement) and (new tape symbol, direction of storage head movement) pairs to the originating cells, which in turn update their own and, if necessary, their neighbors' storage tapes to reflect this move. Thus to simulate a single transition of T, M requires at most n time steps. This completes the proof that an L(n)-space BCA can simulate an $(n \cdot L(n))$-tape bounded Turing acceptor whenever L(n) is measurable.

If L(n) is not measurable, then the above algorithm must be altered since cells in M cannot necessarily mark off length L(n) blocks on their storage tapes. We use a technique due to Savitch [21] to remedy the problem. If each cell were somehow given the value of L(n), then by the above procedure M could

determine whether or not T accepts the input x within storage $n \cdot L(n)$. So M operates as follows. M first assumes $L(n)=1$, and then simulates T, testing whether or not T accepts within n storage. If it does, then M accepts. If T does not accept within storage n, then M next assumes $L(n)=2$ and repeats the process. M proceeds in this manner trying larger and larger values for L(n). If T accepts the input x (within storage $n \cdot L(n)$), then M will eventually find the correct value for L(n) and accept x within storage L(n). If T does not accept x, then M never halts on input x.

Conversely, given an L(n)-space BCA M we can construct an $(n \cdot L(n))$-tape bounded Turing acceptor T which simulates it. For L(n) measurable, T first simulates T' in order to mark off the first L(n) squares of its storage tape, and then successively marks off length L(n) blocks, one for each input symbol. A marker is used in each block to indicate the current position of the corresponding cell's storage head. In addition, the leftmost square of each block stores the state of the corresponding cell's finite control. Initially, the ith block's state is the initial state of M's ith cell.

To simulate one step of M, T systematically scans the non-blank portion of its storage tape from left to right. For each block of L(n) squares that T crosses, it remembers the state stored in the leftmost square and the symbol printed at the marked position. By remembering these (state, symbol) pairs

for the most recently crossed three blocks, T can compute the transition of the cell in M associated with the middle block. T then backs up to that block and changes the state, prints a new symbol at the marked square and moves the mark in the appropriate direction. Thus to simulate a single step of M, T requires $3n \cdot L(n)$ time steps, since each block must be traversed three times during this scan. This completes the proof that an $(n \cdot L(n))$-tape bounded Turing acceptor can simulate an $L(n)$-space BCA whenever $L(n)$ is measurable.

As in the first half of this proof, if $L(n)$ is not measurable, then T cannot mark off n blocks of length $L(n)$ within storage $n \cdot L(n)$. We can use the same technique, however, to again systematically try larger and larger values of $L(n)$. If M accepts the input x, then T will eventually find the correct value for $L(n)$ and accept within storage $n \cdot L(n)$. Otherwise, M does not accept x for any $L(n)$, and the procedure described for T continues forever.//

The following result establishes that there is a hierarchy of memory-augmented BCA computations.

Corollary 2.1. If $L_1(n)$ and $L_2(n)$ are constructable tape functions with

$$\inf_{n \to \infty} \frac{L_1(n)}{L_2(n)} = 0 \quad \text{and} \quad L_2(n) \geq \frac{\log n}{n} \quad ,$$

then there exists a set accepted by an $L_2(n)$-space BCA which is not accepted by any $L_1(n)$-space BCA.

Proof: Stearns, et al. [13] proved that if $L_1(n)$ and $L_2(n)$ are constructable tape functions with

$$\inf_{n\to\infty} \frac{L_1(n)}{L_2(n)} = 0 \quad \text{and} \quad L_2(n) \geq \log n,$$

then there exists a set accepted by an $L_2(n)$-tape bounded Turing acceptor which is not accepted by any $L_1(n)$-tape bounded Turing acceptor. Thus the corollary immediately follows from Theorem 2.1.//

3. Log-space bounded cellular automata

It is well known that BCA's can accept many one- and two-dimensional languages in time proportional to the diameter of the input [3,5-8]. Those algorithms are readily applied to obtain fast algorithms for performing many basic image analysis tasks. For example, image segmentation tasks such as thresholding and, more generally, pixel classification based on point or local properties and a given set of a priori designated classes, can be performed by a BCA since each pixel's classification depends only on local property values. See [22] for a detailed description of thresholding on CLIP4.

Other segmentation techniques such as pattern matching and edge detection also only involve the detection of local properties and so they too are BCA computable in a bounded number of time steps (we are, of course, ignoring the O(diameter) time needed to "broadcast" a given template to each cell in the array). Another technique classifies pixels "fuzzily" by iteratively adjusting each pixel's degree of membership in each of the possible "object" classes according to its local consistency with the memberships of neighboring pixels [23]. This so called relaxation procedure is therefore also computable by a BCA if we assume that the degree-of-membership function has a bounded range and the given compatibility coefficients for pairs of class assignments at neighboring pixels have bounded precision.

In this and the next two sections we investigate memory-augmented BCA's in which each cell has an amount of storage proportional to the logarithm of the input size, i.e., log-space BCA's. Each cell will be considered to have a fixed number of registers, each of size sufficient to store numbers as large as the array. Of course, all of the fast BCA algorithms, such as the segmentation algorithms alluded to above, can be used directly by log-space BCA's which ignore their auxiliary storage. On the other hand, log-space BCA's will be shown to efficiently and conveniently perform many more tasks, in particular those where arithmetic rather than logical operations predominate. This class of memory-augmented BCA's is of practical interest for image analysis since each cell can now store such information as its own coordinates in the image and various measurements of image or region properties.

In one dimension, log-space BCA's can be thought of as two-dimensional BCA's with n columns and log n rows, and, except in the bottom row, cells are connected only to their neighbors above and below them. Figure 3.1 illustrates such a configuration. Similarly, in two dimensions, a log-space BCA is like a stack of log n n-by-n cellular arrays. However, we shall not use this stack model here, but rather shall regard log-space BCA's as arrays of augmented cells.

3.1 Comparisons with other types of acceptors

The results of Section 2 immediately imply corollaries which establish the power of log-space BCA's. From Theorem 2.1 we immediately have for both one- and two-dimensional BCA's

Proposition 3.1. There exists a set accepted by a log-space BCA, but not accepted by any (constant-space) BCA.

A deterministic, nonerasing stack automaton, introduced in [24], has a two-way input tape, a finite control and a stack. The stack may be modified only by adding symbols at the top, i.e., no erasing of symbols is allowed. In addition, the stack head may move up or down the stack in a read-only mode.

Proposition 3.2. The class of sets accepted by log-space BCA's is exactly the class of sets accepted by deterministic, nonerasing stack automata.

Proof: Hopcroft and Ullman [24] proved the equivalence of deterministic, nonerasing stack automata and deterministic (n log n)-tape bounded Turing machines. The proposition now follows from the equivalence of (n log n)-tape bounded Turing machines with log-space BCA's shown in Theorem 2.1. Again it does not matter whether the BCA is one- or two-dimensional.//

3.2 One-dimensional log-space BCA's

In this section we describe log-space BCA algorithms for many image recognition and processing tasks. We begin by considering problems which have one-dimensional analogs in order to simplify the description of these algorithms. In Sections 4 and 5 we describe log-space BCA algorithms for tasks which are inherently two-dimensional.

3.2.1 Local property counting. Smith [6] gives an O(diameter) time procedure due to Meyer for recognition of the majority predicate, i.e., the set of all arrays over input state set {0,1} in which there are more 1's than 0's. That procedure uses a unary to binary conversion technique in order to count the occurrences of each type of input state.

A log-space BCA can use this same technique to count local properties, except that now only a single cell is needed as the accumulator. For example, the area of a subset S of a digital picture is measured as the number of pixels in S. Assume these points are "labeled" at the cells where they occur by being in some particular state, say z. A log-space BCA which computes the area of S and stores it in a specified register, say A, at the leftmost cell is defined as follows. At time step 1 cell 1 sets its A register to 1 if it is in state z, 0 otherwise. Beginning at time step 2 all the z's shift left at unit speed. Each time step that cell 1 receives state z, it increments its A register. After diameter time steps cell 1's A register contains the desired count.

Area is a geometrical property of a region since it does not depend on the gray levels of the specified subset of points. Another class of properties which are commonly used for picture or region description do depend on the gray level distribution of the points. In particular, an important subclass of gray level dependent properties are statistical features based

on the relative frequency with which various local gray level properties occur in a picture or region. For simplicity we consider the former case.

In order to compute these features, a picture must first be mapped into a "property space" of measurements taken at each picture point and which depend only on the gray levels of the point and its neighbors, not on the (global) spatial arrangement of gray levels. Since this class of gray level property density functions depend on a fixed number of neighbors and a quantized gray level range, we can assume that a finite range of property values is sufficient. Hence the domain of the property space is bounded, and the range grows with the picture size. The advantage of log-space BCA's is obvious here where a fixed set of registers can accumulate the measured values from every point. Examples of such property spaces include histograms and cooccurrence matrices.

The general procedure for computing such features on a log-space BCA is as follows. First, each cell computes in parallel its local property value in a bounded number of time steps. Afterwards, each cell routes its value to a designated cell which counts the number of occurrences of each property value. This requires diameter time steps. Finally, the designated cell computes a specified feature based on the distribution of property values in the property space.

The gray level histogram of a digital picture quantized to k levels is a vector H, where $H(z_i)$ is the number of points in the picture with gray level z_i. This gray level frequency vector is computed by the leftmost cell in a log-space BCA as follows. At time step 1 cell 1 initializes k registers, $H_1,\ldots,H_k$, to 0. Beginning at time step 2 the entire picture shifts left at unit speed. If cell 1 receives gray level z_i, then it increments register H_i. After diameter time steps cell 1 contains the histogram of the picture.

Statistical properties of a picture's histogram are easily computed with a log-space BCA. We will assume that the precision of these feature values grows at most linearly with the input size so that they can be stored at a single cell. For example, techniques for finding valley bottoms (which are often reasonable points at which to threshold a picture) or p-tiles can be directly implemented by the cell storing the histogram. Details will not be given. The mean and variance of a picture's gray levels can also be computed directly from the histogram since the arithmetic operations involved require storing intermediate quantities which are bounded by a fixed multiple of the picture size.

Similarly, the gray level cooccurrence matrix of a picture, which measures how often each pair of gray levels occur at a specified relative displacement, can be computed. Cell 1 must

now store k^2 registers, where again k is the number of gray levels. The algorithm only differs from the previous one in the initial property measurement, which requires a copy of the picture to be shifted the specified distance, but in the opposite direction, given by the relative displacement of pixels to be compared. Afterwards, these gray level pairs commence shifting leftward and are counted by the leftmost cell.

3.2.2 Moments. A second subclass of gray level properties do depend on the spatial arrangement of gray levels in the picture. The moments of a picture are examples of such properties. They are often useful as measures of location, shape, and for geometrical normalization. The ith moment of a one-dimensional picture f is defined to be $m_i = \sum_x x^i f(x)$.

The coordinate of a picture's centroid is given by m_1/m_0. The pixel closest to a picture's centroid can be marked by a log-space BCA in O(diameter) time steps as follows. Let each cell have two registers, A and B. At time step 1 each cell with input gray level (i.e., state) z_i sets its A and B registers to i. Beginning at time step 2 m_0 is computed by cell 1 by modifying the histogramming procedure so that when gray level z_i arrives, cell 1 adds i to the contents of register A. Assuming these additions take unit time, this phase is completed in diameter time steps.

At the next time step cell 1 divides the contents of its A register by two and then subtracts its gray level, stored in its B register, and enters state p. This count is then propagated rightwards at one-half unit speed. That is, if a cell's left neighbor is in state p, then the current cell copies its left neighbor's A register into its own A register and enters state q. A cell in state q subtracts its B register from its A register and stores the result in its A register. At the

same time the cell tests whether or not the new count is less than or equal to zero. If it is not then the cell enters state p, otherwise it enters state r. The cell which enters state r is at the picture's "center of gravity" since half of the picture's gray levels are on either side of this point. Thus this cell is the centroid of the picture. In the worst case the centroid is the rightmost pixel in the picture, so the complete procedure takes at most three times diameter time steps.

If we use the centroid as the origin, the moment of inertia, m_2, is an important descriptor since it is a rotation- and translation-invariant property. We now describe how a log-space BCA M can compute m_2 and store it at the centroid in O(diameter) time. Assume each cell has two registers, A and B. Using the procedure described above M can find the centroid and set each cell's B register to its input gray level within three times diameter time steps. When the centroid enters state r it initiates a unit speed signal sent to its left and right, along with an incrementing counter so that each cell computes its position in the new coordinate system. Specifically, when the centroid cell enters state r it simultaneously sets its A register to zero. All other cells act as follows. The first time that one of a cell's neighbors is in state r or s, then the current cell copies that cell's A register into its own A register, increments it, and enters state s for one time step. In this way each cell stores in its A register its distance from

the origin. If we assume multiplication takes unit time, then each cell can now compute $x^2 \cdot f(x)$ in two more time steps. Hence after at most diameter+2 time steps each cell has this quantity stored in its A register. If each cell remembers in which direction the origin is located, then in no more than diameter time these numbers can be shifted to, and summed by, the cell at the centroid.

3.2.3 Autocorrelation. The autocorrelation of a one-dimensional picture f is defined as $R_f(x')=\sum_x f(x)f(x-x')$. We now describe how a log-space BCA can compute R_f in $O(\text{diameter}^2)$ time by shifting a copy of the picture with respect to the original, and summing the pairwise products of coincident gray levels at each relative displacement.

Let each cell have three registers, A, B, and C. At time step 1 if a cell's input gray level is z_i, then it sets its A and B registers to i. At the next time step each cell multiplies the contents of its A and B registers, storing the result in its C register. For the next diameter-1 time steps the contents of these C registers shift leftward, and are summed in cell 1's C register. (Since this sum is no larger than $\text{diameter}\cdot k^2$, there is no problem storing it in a single register.) When cell 1 adds the value sent from the rightmost cell, its C register contains the value of $R_f(0)$, so it initiates a firing squad which starts the computation of $R_f(1)$. First, the picture shifts right one position by having each cell simultaneously copy the contents of its left neighbor's B register into its own B register. We assume that the picture is zero outside of its given domain, so at the next time step cells 2 through n, where n is the length of the picture, multiply the contents of their A and B registers, storing the result in C. Then for the next diameter-2 time steps the C registers

again shift leftward and are summed in cell 2's C register. This process continues until $R_f(n-1)$ is defined in cell n's C register.

The computation of $R_f(i)$ requires 1 time step to shift the picture to the new position, 1 time step to pairwise multiply the cooccuring gray levels, and n-i time steps to shift and sum these values. We have not included the timing for the firing squad synchronization since in fact each cell has enough storage to contain a clock which counts to n-i+2 and therefore no firing squad is necessary. Thus the algorithm takes $\sum_{i=0}^{n-1} n-i+2 = \frac{n(n+1)}{2}$ time steps, i.e., O(diameter^2), to compute R_f.

4. Two-dimensional log-space BCA's for picture representation

Segmentation involves extracting distinguished sets of pixels from a picture but does not, in general, consider how these points are to be grouped so that they can be identified with meaningful objects or regions in the picture. This section considers methods of decomposing such a picture subset into connected regions and then considers data structures for representing these regions. First, we present some definitions and terminology concerning digital topology. See [25] for a complete review.

Given a point p in a digital picture, its four horizontal and vertical neighbors are called its 4-neighbors, and are said to be 4-adjacent to p. These four neighbors together with p's four diagonal neighbors are called p's 8-neighbors, which are each 8-adjacent to p. A 4-path (8-path) from p to q is a sequence of picture points $p=p_0,p_1,\ldots,p_n=q$ such that p_i is 4-adjacent (8-adjacent) to p_{i-1} for all $1\leq i\leq n$. A (4- or 8-) path is called a (4- or 8-) geodesic if no shorter path with the same endpoints exists. Given a picture subset S, we say that p is (4- or 8-) connected in S to q if there is a (4- or 8-) path from p to q consisting entirely of points in S. The equivalence classes under the equivalence relation "connected in S" are called the connected components, or regions, of S.

The subset of picture points not contained in a specified subset S is denoted by $\bar{S}$. $\bar{S}$ can also be decomposed into connected components. If we assume that the picture is embedded in a larger picture consisting only of points in $\bar{S}$, then exactly one of $\bar{S}$'s components contains this set of "boundary points." This region is called the background of $\bar{S}$ and all other components of S are called holes in S. If S has no holes, it is called simply-connected. The border of S is the set of points of S that have at least one neighbor in $\bar{S}$. If p is in the border of S, then we call p a border point of S. More specifically, those border points which are adjacent to the background are on the outer border of S, the other points are on hole borders of S. The set of points of S which are not on the border of S are called interior points of S.

4.1 Region counting and labeling

Beyer [3] and Levialdi [26] describe a method for counting the number of connected components in a binary picture by a BCA in O(diameter) time. Their method shrinks in parallel each connected component of 0's or 1's to a single point, the upper left corner of the component's upright framing rectangle, without disconnecting or merging components in the process. Briefly, the topology-preserving shrinking transformation ϕ is a two time step operation over input state set {0,1} that

1. changes a cell in state 1 to state 0 if its right and lower neighbors are both in state 0,
2. changes a cell in state 0 to state 1 if its right, lower, and lower-right diagonal neighbors are all in state 1, and
3. otherwise, a cell remains in the same state.

See [3,25,26] for the proof that applying ϕ preserves connectedness of both the 1's (which are 4-connected) and the 0's (which are 8-connected).

We now consider the problem of assigning distinct labels to the connected components of a picture subset S, i.e., all pixels in the same component are given the same label, but no two pixels in different components may have the same label. Since there may be an unbounded number of components in a picture, a BCA does not have enough states to distinctly label all of the components. (Consider the checkerboard picture which has O(area) 4-components of one pixel each. Each cell

has a bounded number of states to store its label, but an unbounded number of labels are needed.) Thus this problem is only meaningful on a log-space BCA where enough storage is available for label names.

To solve the problem, we divide it into the following subtasks. First, a distinguished cell in each region is located. This cell generates a unique label for the component it represents, and then broadcasts the label back to the other cells in the region. We assume the input state set is {0,1} and we wish to label all 4-connected components of 1's in the picture.

a) Distinguished cell marking. Smith [6] gives a BCA algorithm which finds the rightmost cell in the uppermost row of a region in time proportional to the perimeter of the region. That procedure constrained the computation to the border cells of the region. Since in the worst case perimeter time equals area time, we now describe an alternative algorithm which uses cells outside the region to find the uppermost, leftmost cell in the region in time proportional to the diameter of the region's upright framing rectangle. This achieves the lower bound time for distinguished cell marking since this is the amount of time needed for a single cell to "see" the entire component.

The algorithm is a modification of the Beyer/Levialdi algorithm; here we shrink a copy of a component to a single

point and then route a message rightward until it meets the original component at its uppermost, leftmost point. First, notice that the message routing cannot be done by a BCA which simply marks the leftmost 1-cell that the message meets. For example, Figure 4.1 shows a situation where region A shrinks to the point a which would first meet region B on its trip back towards A.

We now describe the actions of a log-space BCA M which marks the uppermost, leftmost cell in a connected component of 1's, S. First, each cell stores its input state in its G register and then computes its matrix coordinates, storing its row coordinate in its R register and column coordinate in its C register. Specifically, at time step 1 each cell sets its R and C registers to 0, the upper-left corner cell enters state p, and all other cells enter state q. Beginning at the next time step, if a cell c is in state q and either its left or upper neighbor is in state p, then c copies that neighbor's (if both are in state p then pick the left one) R and C registers into its own and increments one of its registers as follows: if c copied its upper neighbor's registers, then it increments its R register; otherwise, c increments its C register. Cell c then enters state p. Clearly, after diameter time steps every cell has computed its coordinates. The next phase is initiated by the lower-right corner cell which, when it enters state p, starts a firing squad. (Again, the firing squad could be avoided since each cell can be made to count diameter time steps after initially storing the array's diameter in a

counter in each cell.)

At the first time step of this next phase each cell enters the state stored in its G register and those candidate upper-left corner cells in S (i.e., a cell in state 1 whose upper and left neighbors are in state 0) store a copy of their coordinates in two other registers, A and B. We now modify ϕ so that when a cell in state 0 changes to state 1, it also copies the contents of its right neighbor's A and B registers into its own A and B registers. When this modified Beyer/ Levialdi algorithm shrinks S to a single point, we claim that this cell's A and B registers contain the coordinates of S's uppermost, leftmost point. Therefore, when a 1-cell finds it is an isolated point, it then routes a message rightward to this designated cell.

We will now show that the isolated cell's A and B registers contain the stated coordinates. Given S, consider its uppermost, leftmost cell c. If c is an isolated point, then clearly c's A and B registers contain the appropriate information. Otherwise, at least one of c's lower or right neighbors is also in state 1. In either case c will not change to state 0 during the first application of ϕ. Indeed c will not change states until both c's lower and right neighbors are in state 0. If all of S is in the same column or to the right of c, then c is already in the upper-left corner of S's framing rectangle, so again c's A and B registers will be correct, since c will remain in state 1 until it is an isolated point.

Otherwise, S extends into the column to the left of c's column. This and the fact that ϕ preserves S's connectedness imply that c will still be in state 1 when eventually both c's lower and lower-left corner neighbors are in state 1. c's left neighbor must still be in state 0, so at the next application of ϕ c's left neighbor enters state 1 and copies c's A and B registers. At some later time step c enters state 0. The same arguments can now be made concerning c's left neighbor, i.e., that this cell will not enter state 0 before its left neighbor enters state 1 and copies c's coordinates from the cell. Thus c's address is passed from right to left along the top row of S's framing rectangle, eventually being copied by the upper-left corner cell of this rectangle, call it d, in advance of the vanishing component.

Cell d now routes a signal back to cell c as follows. During the first time step cell d tests whether or not the contents of its C register is equal to the contents of its B register. If they are equal, then the destination cell is d itself. Otherwise, cell d enters state r for one time step. When a cell's left neighbor is in state r then the current cell copies that cell's B register into its own B register and enters state s. In state s the cell compares its B and C registers; if they are equal then this cell is c, otherwise it enters state r for one step. Thus the column coordinate moves right at 1/2 unit speed until it arrives at cell c, the uppermost, leftmost cell in S. (Note: this procedure is easily

generalized so that a cell can send a message to any other specified cell in the array. In this case both the A and B registers containing the address of the destination cell must be passed from cell to cell. In addition, message registers can be passed, perhaps containing the sender's coordinates so that a reply message can be sent.)

The coordinate labeling phase of this algorithm requires diameter time steps, firing squad synchronization takes twice diameter time [27], the modified Beyer/Levialdi algorithm takes 2(h+w-2) time, where h and w are the height and width of S's upright framing rectangle, and the final designator signal takes at most 2w time steps. If we assume coordinate labeling and the subsequent synchronization process are pre-processing steps, then the uppermost, leftmost cell in S can be so designated in at most four times diameter time.

This algorithm cannot be used, however, to simultaneously find the uppermost, leftmost point in every connected component in a picture. The shrinking stage was shown by Beyer and Levialdi to perform properly in multiple region shrinking because it has the properties of never merging connected regions, and each region vanishing at different cells or different times. The final message routing procedure, on the other hand, does not guarantee that an unbounded number of messages won't conflict with one another during their travel. Figure 4.2 shows an example of the problem, each region

collapsing into its upper-left corner just as all the messages from regions above it are moving past this cell toward their designated cells.

b) Label generation. Each distinguished cell in a picture must now generate a unique label for the component it represents. The simplest way is for each cell to use its coordinates as the component's label. We have shown above how each cell can compute its matrix coordinates and store them in a pair of registers in diameter time steps.

c) Pixel labeling. Each distinguished cell must now broadcast the label to all cells in the component. We do this phase by generalizing the coordinate computation technique so that the label is passed only to neighbors which are in the component. The ordered pairs of neighboring cells in which labels are passed defines a set of directed arcs from the distinguished cell to every cell in the component. Furthermore, these arcs define minimum length paths from the given cell to every other. Thus in addition to the labeling process, subsequent procedures may have use for these minimum time propagation channels. We now formalize the definition and construction of this rooted minimum spanning tree (MST). This structure makes no demand on the augmented memory, and so can also be done by a BCA.

Given a specified point c in a 4-connected region S, define the minimum spanning tree of S rooted at c to be a rooted

directed acyclic graph in which the vertices are the points of S and arcs exist between horizontally or vertically adjacent vertices such that the following properties are satisfied:

1. Vertex c is the root, i.e., there are no arcs directed from c to any of its neighbors.
2. Every vertex except the root has exactly one arc which is directed from it.
3. Every vertex u is connected to the root c by a unique path $(v_1,v_2),(v_2,v_3),\ldots,(v_{n-1},v_n)$, where $u=v_1$, $c=v_n$, and (v_i,v_{i+1}) is the arc directed from v_i, for all $1 \leq i \leq n-1$.
4. The path associated with each vertex u to the root is of minimum length, i.e., there does not exist another choice of arcs between horizontally or vertically adjacent vertices (of S) which satisfies properties 1-3 and results in a shorter sequence of arcs from u to c.

Figure 4.3 shows an example of a region and a rooted minimum spanning tree for one of its points.

We now show how a BCA with a distinguished cell c_0 in a component S can construct its MST. The algorithm, based on a similar one by Beyer [3], takes O(intrinsic diameter of S) time. Initially, we assume each cell in S is in state 1 and its A register is set to zero, and all other cells are in state 0. At time step 1 cell c_o enters state p for one time step and

then enters state q. Every other cell in state 1 remains in state 1 until at least one of its neighbors enters state p. At the next time step the cell enters state p for one time step before entering state q, setting its A register to either 1,2,3 or 4 depending on which neighbor was in state p (in case of multiple p-neighbors, choose one according to the precedence, say, upper, left, lower, right). In this way a signal p propagates from cell to cell through S, each cell in S entering state p after a number of time steps equal to its distance through cells in S from c_o. Thus the chain of direction links, stored in each cell's A register, from a cell back to c_o is a minimum length path.

If when a cell is in state p all of its four neighbors are either in state 0 or r, then the current cell entered state p at the same time or later than all of its neighbors in S. Designate these cells which are a local maximum distance from c_o as the <u>leaf</u> vertices in c_o's MST. At the time step after a leaf cell enters state p it enters state r. Each cell in state q remains in that state until all of its sons (i.e., neighbors with links directed back to it) enter state r; then it enters state r. Thus the r signals are initiated by the leaf cells and propagate back up the tree signaling the completion of the algorithm.

In general, this return signal may also contain a description of the subtree rooted at the cell in state r. For example,

each cell can compute the height of its subtree as follows. Each leaf cell sets its H register to zero. When a cell enters state r it also copies that son's H register which has the maximum value into its own H register and increments it.

In the remainder of this section we consider alternative representations for a given labeled connected component, determining how fast a log-space BCA can transform an array representation to each alternative, and vice versa.

4.2 Run length coding. Given a region S, each row of the picture consists, in general, of runs of pixels in S separated by runs of pixels in $\overline{S}$. Thus we can represent S by a list of run (starting position, length) pairs. If the runs, on the average, are sufficiently long, then this representation is more compact than, and yet an exact coding of, the original array representation of S. Since S is connected it cannot skip rows, and therefore we can shorten the coding further as follows. A run length coding of S consists of a starting row "header message," followed by run (starting column, length) pairs, with a punctuation bit separating runs on adjacent rows.

A log-space BCA M can output this run length coding of a connected component S at a designated cell, say the accept cell, as follows. Each cell needs three registers R, C, and L, all initially set to zero. Assume cells in S are initially in state 1, all others are in state 0. At time step 1 cells at the left and right ends mark themselves. Beginning at time step 2 each right end cell initiates a signal which is sent to the left end of the run. In conjunction with this each left end cell increments its L register at each step until the signal arrives at the cell. Thus when the signal arrives each run's length (minus one) is stored at the left end cell of the run.

Next, each row's runs are packed in order at the left end of the row. That is, if a row has k runs, then the ith run's

length register L shifts left as far as it can, stopping at the ith cell from the left end. At the same time the C register associated with each left end cell is shifted left together with the run's L register, each intermediate cell incrementing the C register as it passes.

M now outputs this set of runs in row-major order as follows. The leftmost run in the top row of S marks itself as the first run after detecting that the cell above it contains no run description. At the next step this run description shifts up the left column to the accept cell, each intermediate cell incrementing this run's R register so that when the run is output, R contains the starting row of S. The other runs in the top row shift left to the first column and then up to the output cell, following immediately behind the run descriptions ahead of them. When a run description (i.e., a C,L register pair) shifts left, again the C register is incremented; thus when each run turns up the first column its C register contains the run's starting column.

The first run in every other row waits until the cell above it has shifted out all of its row's run descriptions. Then this run inserts a new row punctuation mark between it and the last run of the previous row before it commences to move up the left column to the output cell. In the worst case there can be O(area) runs in S, requiring O(area) time to output this representation. (In such situations, however,

run length coding would not be the appropriate representation anyway.)

Reconstruction of S from this representation is straightforward. The first run of S, containing the starting row of S, shifts down the first column, its R register being decremented and tested for zero by each cell that receives it. When the R register is zero, the current cell marks itself as the starting row. The (starting column, length) pairs shift down the first column to the marked cell, then the column register C is decremented as the pair of registers shift right to the run's starting position. When the C register is zero, the length register L continues shifting right. Each cell that now receives a nonzero L register marks itself in S, decrements the L register, and sends it on to its right. When a punctuation bit travels down the left column and arrives at the marked current row, that cell's mark is erased and the mark rewritten at the cell below it.

4.3 Chain coding. If a region is relatively compact then its perimeter is proportional to the square root of its area. This implies that such regions can be efficiently stored by saving a description of their border points only. Trivially, a BCA can mark a region's border in one time step.

Given a starting border point and an adjacent background point, we can traverse these border points by following the border while always "keeping our right hand" on S [9]. In fact the direction of traversal of a given border through a given border point by this sequential border following rule is locally computable in a 3 by 3 neighborhood of the point. Thus each border point can determine, for each border passing through it, its predecessor and successor border points. If we represent each successor point of a border point by an octal code, where the correspondence between digits and neighbors is $\begin{matrix}321\\4*0\\567\end{matrix}$, then the border of S can be described exactly by specifying the coordinates of a starting point and an ordered sequence of octal digits. Furthermore, the "right hand rule" implies that the inside of S's border is always determined by the link direction, so no other information is needed to represent S.

To output the chain code of a region S at a designated cell it is first necessary to find a distinguished starting point for each border of S. S and each connected component in $\overline{S}$ can locate its uppermost, leftmost point in time proportional to the component's diameter using the procedure in Section 4.1. Given a component in $\overline{S}$ with designated cell c

on its border, specify one of c's S-neighbors (if any exist), as the designated starting cell of this hole border of S.

Assume that each cell has previously computed its coordinates, and that associated with each starting border point's link are its coordinates. To output the chain code of S at designated cell c, the starting cell of the outer border, d, first establishes an output path to c using the message routing technique described in Section 4.1. Each cell along this directed path and the directed path of outer border cells now acts in "bucket-brigade" fashion, passing the chain code link by link clockwise around the border to d and then along the output path to c. A hole border's chain code is passed counterclockwise around its border to its start cell. From this cell the links are passed up its column until they hit another border of S. Here the procession of links waits until that border's code has passed, and then follows it in the same direction around its border and eventually to the output cell c. Thus hole border codes "bubble up" around other holes above them until they reach the outer border. Consequently, to output S's chain code requires O(perimeter of S) time steps, where perimeter is the number of border points in S. Notice that log space is used only to store the borders' starting coordinates, not the chain codes themselves.

To reconstruct S, the first link, containing the coordinates of S's outer border's starting point, establishes a path back to the starting cell. The remaining links of the chain code

follow immediately behind and reidentify border cells from their link types after a counterclockwise traversal of the partially reconstructed border of S back to the link's successor cell. The direction of the link also defines which side is the interior of S. A link which starts a new hole border departs from the outer border at cell d and heads directly for its starting location. From there, the border is reconstructed in a clockwise order around the hole.

We can relabel the interior points of S since each border point knows which direction is the inside of S, and every run of S is bounded by a pair of border points. Hence each border cell can initiate a signal to cells in the proper direction in its row, marking them as part of S until it meets another border point or an opposing relabeling signal. Since these signals are stopped by the opposing border point in its run, it is necessary to wait until the border has been completely reconstructed before commencing this process. If S has no holes, then the starting cell d sends a signal around to every border point of S after the final link has been reconstructed. Otherwise, each hole border's starting cell waits until its border has been completed, and then sends a completion message to the outer border's starting cell. When the outer border's starting cell has received completion messages from all of its hole borders, it commences the relabelling process as follows. The outer border points are signalled

directly from the starting cell, which sends a message through the border. These cells then relabel points until they hit the opposing border. If the relabeling signal hits a hole border, then this initiates a signal through all cells of this border to begin the relabeling process. In this way runs of S which are bounded on both sides by hole borders are eventually filled in as desired.

4.4 Skeletons. Given a connected component S of cells in state 1 and all other cells in state 0, we can associate with each point of S its distance to the closest point of $\bar{S}$. This distance transformation of S is readily computed by a log-space BCA M with a single register D at each cell as follows. Beginning at time step 1 each cell increments its D register at each time step as long as the cell is in state 1. At time step 1 each cell in state 1 which has at least one of its neighbors in state 0, enters state 2. Subsequently any cell in state 1 with at least one of its neighbors in state 2 enters state 2. Thus at each step all border points of S are changed to 2's, so that S is successively "thinned" until it disappears. Each cell's D register counts how long it takes for the point to be deleted from S, and thus contains its distance from $\bar{S}$.

The set of points in S whose distances from $\bar{S}$ are local maxima (i.e., no neighboring point has greater distance from $\bar{S}$) defines the medial axis of S. This set is easily computed from the distance transform in four time steps by comparing each cell's D register with its four neighbors' D registers. Now if we associate with each of these points its distance from $\bar{S}$, we obtain an exact representation of S called its medial axis transformation. Thus diameter + 4 time steps are required in the worst case for M to compute the medial axis transformation for all regions in a picture.

Reconstruction of a region S from its medial axis transformation is accomplished by reversing the skeletonization procedure so that border points are added at each step and S grows back to its original size. More concretely, assume each cell in the medial axis transformation of S is in state 1 and all other cells are in state 0. At every time step each cell which is in state 0 and has at least one of its neighbors in state 1, copies the D register of one of these neighbors and enters state 2. If a cell is in state 2 and its D register is 1, then at the next step it enters state 3. Otherwise, the cell decrements its D register and enters state 1. After diameter time all cells in S are in state 1, all cells in $\overline{S}$ which are 4-adjacent to S are in state 3, and all other cells in $\overline{S}$ are in state 0.

In general, the medial axis of a connected component is not connected. If desired, we could output the connected chains of "skeletal" points at a designated cell, say the accept cell, as follows. Assume each cell has previously computed its coordinates and a distinguished cell in each connected chain has been marked using the techniques in Section 4.1. We can represent a connected skeleton by a depth-first chain code traversal of its skeletal points from its starting cell. That is, each skeletal point can have up to three neighboring skeletal points; hence from a designated starting point a skeleton is a binary tree of skeletal points connected by implicit edges between 4-adjacent points, with

the designated cell as the root.

The root can output its skeleton's chain code in a depth-first order as follows. The root starts a signal at time step 1 which traverses the tree depth-first. When the signal moves to a point for the first time it computes the chain link corresponding to that move and the degree of the node in the tree (i.e., the number of skeletal neighbors). This (link, node type) pair then begins "bubbling up" the tree by following the path taken by the link immediately ahead of it. When the downward moving signal arrives at a node of degree three, it chooses an unmarked son node, marks it, and continues as before. At a leaf node (i.e., degree equals one), the signal reverses direction and follows the chain of links it just generated back to the previous node of degree three. Here, if there is a remaining unmarked son, the signal marks it and starts down that subtree, again computing links which follow immediately behind the links from the previous subtree. If there are no unmarked sons at a node of degree three, then the signal continues backtracking up the tree. This process takes time equal to twice the number of skeletal points since each point is visited twice by the traversing signal. To output this chain code at the accept cell now only requires the root to mark a path along which the chain can follow.

Other algorithms exist which "thin" a region into a set of arcs, but do preserve connectedness. These algorithms

are also parallel and locally computable and so can be performed by an ordinary BCA. Outputting the chain code of this set of connected points can be done using the technique described above. The region cannot, however, be reconstructed from this representation because of the special conditions needed to preserve connectivity and avoid deleting end points.

4.5 Quadtrees. Consider a sequence of repeated subdivisions of a picture into quadrants, so that at the kth subdivision the picture is partitioned into 2^k by 2^k grid squares. Given a 2^n by 2^n picture, this defines a tree of degree 4 in which the nodes represent grid squares, the sons of a node are its four quadrants, and the root is the original picture. A region S in the picture can be represented, to any desired degree of accuracy, by the union of maximal grid squares (of sizes and positions specified by the picture's recursive partitioning) that are contained in S. In general, this areal representation provides an efficient description of relatively compact regions, since in this case large numbers of pixels can often be represented by a single leaf node in the quadtree. In another way, a quadtree can be thought of as a generalized MAT in which only a specific set of square disks, whose sizes and positions are powers of two, are allowed.

A log-space BCA can compute the quadtree representation of a region S, storing each node at the center of its 2^k by 2^k grid square, as follows. The construction of the quadtree will be bottom up, the center cell of each grid square at level k routing information about its base's contents diagonally across the array to the center of the 2^{k+1} by 2^{k+1} square of which it is a quadrant. A node at level k+1 then computes whether or not it is in S's quadtree from the infor-

mation passed from its four sons: If all four sons are leaf nodes in the quadtree, then the current node must delete its sons from the tree and insert itself as a leaf since its entire base is in S. If no sons are in the quadtree, then S does not intersect the current node's base at all so it should not be part of the quadtree either. Otherwise, the current node inserts itself as a nonleaf node in the tree since points in both S and $\overline{S}$ are in its base. This process continues until the root of the quadtree is determined after no more than diameter time steps.

More specifically, assume that the input picture is size 2^n by 2^n, and each cell contains five registers, G, R, C, T, and L. Let register G contain 1 if the cell is in S, 0 otherwise. Furthermore, let [X] denote the contents of register X. In diameter time steps each cell computes its n-bit row and column coordinates, $r_n \ldots r_2 r_1$, and $c_n \ldots c_2 c_1$, and stores them in its R and C registers, respectively, using the procedure described in Section 4.1. Simultaneously, each cell sets its T register to 0 and L register to 1. At the next time step each cell sets the first bit in its T register to 1 if the pixel at that cell is in S. Next, each cell determines the direction of its father cell from the least significant bits in its address. That is, a cell's least significant column bit, c_1, equals zero if it is in an even numbered column (remember, the upper left cell is at position (0,0)), so c_1=0 implies that the cell's father is in a column to its

right. Similarly, testing the value of a cell's least significant row bit indicates whether its father is in a row above or below its own row. Thus combining the information from these two bits indicates which of four possible diagonal directions to move in order to find one's father in the quadtree. Beginning at the next time step, each cell sends in the indicated direction the least significant bit in its T register, i.e., whether or not the cell is in S. We will define the center of a 2^k by 2^k block of cells to be the upper-left corner cell in the 2 by 2 block of cells which surround the center point. Thus at the end of two time steps, the upper left corner cell of each 2 by 2 block receives the messages from its sons' quadrants.

If all four sons are in S, then the current cell becomes a leaf node at level 1 of S's quadtree and deletes its sons at level 0. This is done by incrementing L, setting the [L]th bit in T, $t_{[L]}$, to 1, and returning a message to its four sons to change the least significant bits in their T registers to zero. If none of a node's four sons are in S, then the cell increments L and sets $t_{[L]}$ to zero. Otherwise, only some of the current node's sons are in S, so the cell makes this node a non-leaf node in S's quadtree by incrementing L and setting $t_{[L]}$ to two.

In general, the description of S's quadtree at level k is stored in the kth bit of the T registers of the center cells in each 2^k by 2^k block. The nodes at level k+1 are computed

after each cell representing a node at level k routes a message about its node type (t_k=0 indicates the node is not in S's quadtree, 1 implies it is a leaf node, and 2 a nonleaf) to its father node. Since sons don't know the address of their father, they can only send the required information in the proper direction. The L register is used to store the current level number, and $r_{[L]}$ and $c_{[L]}$ determine in which direction to send the current node's description. When four messages meet 2^k time steps later, the cell they intersect at is precisely the cell which is their father node (Figure 4.4). Each cell representing a node at level k+1 then computes its node type, and routes a copy of this information on towards its father. If this node is a leaf, then a reply message is returned to each son, which then deletes the node from the quadtree. In the worst case, the root of S's quadtree is at level n, which is computed 2^n time steps after level 1 nodes are determined. Thus the quadtree representation of a region is computable in O(diameter) time.

To reconstruct a region from its quadtree, traverse the tree top down in parallel until a leaf node is reached. This node then relabels all of the cells in its base. The traversal and relabelling are most easily implemented if a cell first computes the coordinates of its sons and of the four corner cells in its base, respectively.

The techniques used to generate a region's quadtree can be modified to construct reduced resolution versions of a picture by repeated 2 by 2 averaging. In this case we construct a complete quadtree for the entire picture, each node in the tree storing the average gray level of the pixels in its base. This gray level is computed by averaging the four gray levels sent to the node from its four sons.

5. Two-dimensional log-space BCA's for picture description

After a picture has been segmented into regions, it often is desirable to obtain a description in terms of properties of the picture and its parts, and the relationships among these parts. Section 3.2 described methods by which log-space BCA's can measure certain gray-level properties of one-dimensional pictures. Those techniques can be generalized to two dimensions and so will not be discussed further. In this section we investigate the ability of log-space BCA's to measure geometrical properties of a region in a two-dimensional picture. Geometrical properties, unlike gray level properties, depend only on which points of the picture belong to the region, not on the gray levels of these points. While it has been shown that BCA's can perform counting operations, e.g., in finding a distinguished cell [6] or counting the number of pixels with a specified label [8], log-space BCA's perform such operations more naturally. Other properties such as autocorrelation can only be measured by log-space BCA's, since a BCA does not have enough memory to even store the property. In this section we describe how log-space BCA's can measure the geometrical region properties area, perimeter, compactness, elongatedness, width, height, diameter, and convexity. We begin by reviewing some basic concepts of distance and diameter in digital pictures.

Given two points $p=(x,y)$ and $q=(u,v)$, define their city-block distance as $d_4(p,q)=|x-u|+|y-v|$, and their chessboard distance as $d_8(p,q)=\max(|x-u|,|y-v|)$. For simplicity, from now on we only present definitions which result from using city-block distance and 4-connectedness. An analogous set of definitions also exist using chessboard distance. Given a 4-connected component S and p,q in S, it can be shown [9] that $d_4(p,q)$ is just the length of a shortest 4-path from p to q through points in the picture. If we restrict the path of points to be entirely contained in S, then the length of a shortest such path is called the intrinsic distance between p and q.

The 4-diameter of S is defined as the greatest city-block distance between any pair of points of S. By the intrinsic 4-diameter of S we mean the greatest intrinsic distance between any pair of points of S.

5.1 Area

The area of a region in a digital picture is defined to be the number of pixels in the region. Smith [6] gives an O(diameter) time BCA algorithm for determining whether or not there are more 1's than 0's in a binary picture, in which 1's are counted in each row and then these row counts are summed. That algorithm is easily modified to compute the area of a region in O(diameter) time.

In contrast, a log-space BCA M can compute the area of a region S in O(diameter of S) time using the modified Beyer/ Levialdi algorithm of Section 4.1 to merge the pixels together as fast as possible. That is, each cell in S initializes its A register to 1 at time step 1, and each cell in $\overline{S}$ initializes its A register to 0. M starts the shrinking algorithm at time step 2. Whenever a cell in S is deleted, one of its upper or left neighbors which is in S (and will not be removed by the connectedness criterion) adds the contents of the deleted cell's A register to its own A register. Clearly, when S is reduced to a single point, that cell's A register contains the number of pixels in S. Simultaneously, but not interfering with this process, the coordinates of the uppermost, leftmost cell are computed as described in Section 4.1. Thus the A register can now be sent to the uppermost, leftmost cell of S. Shrinking takes O(diameter of S) time steps and the final message routing process at most width of S time; thus S's area is computed by a log-space BCA in O(diameter of S) time.

Both of these algorithms make use of cells outside of S in order to attain the lower bound time results. However, as pointed out in Section 4, this may cause problems if we want to simultaneously compute the areas of all regions in a picture. Therefore we now present an alternative, based on the minimum spanning tree of a region, which is restricted to those cells in S and hence can be used to compute in parallel all regions' areas. Under this restriction, however, O(intrinsic diameter of S) time is a lower bound.

A log-space BCA can compute the area of a region S and store it at a designated cell in O(intrinsic diameter of S) time steps as follows. First, the uppermost, leftmost point c of S is located using the algorithm in Section 4.1. Next, the minimum spanning tree rooted at c is constructed, defining a unique path from each cell in S to c. During this procedure each cell in S also initializes its A register to 1. Each leaf cell in the tree initiates a reply signal r which propagates back to c along the path indicated by the direction links stored in the cells, each cell adding the contents of its sons' A registers to its own A register. That is, if cell c_1 is in state r, its neighbor c_2 enters state r only if c_2 is the neighbor of c_1 in the direction that c_1 has stored as its arc in the rooted tree. When c_2 enters state r, at the next step c_1 enters state t. Simultaneously, c_2 adds the contents of its own A register and stores the result in its own A register. If c_2 has two or more neighbors for which it

is the stored direction neighbor, then it enters state r only after r has arrived at all of its sons. As each r arrives, c_2 adds the contents of that son's A register into its own, so that when c_2 finally enters state r, its A register contains the number of cells in its subtree. In particular, after a number of time steps at most equal to the intrinsic 4-diameter of S, c enters state r and its A register contains the number of pixels in S.

5.2 Perimeter

The perimeter of a region S can be defined as the number of its border points, or as the total length of its borders' chain codes. In either case, Section 4.3 showed how a BCA can detect these points and compute the chain links in two time steps by looking in a 3 by 3 neighborhood around each point. The log-space BCA algorithm which output the chain code at a designated cell can also be readily adapted to count the number of links or border points as they are output. That is, the output cell contains a special register, initialized to 0, which is incremented each time a link is output. (Note: if desired we can add $\sqrt{2}$ for diagonal links.) Since this process sequentially propagates the links around the border, it requires O(perimeter of S) time to compute S's perimeter.

Alternatively, we now describe a log-space BCA M which computes S's perimeter in O(diameter of S) time steps. During the first time step each cell determines whether or not it is a border point of S, setting its A register to 1 if it is a border point, 0 if it is not. (Or, if desired, each border point computes its link-length contribution to the chain code of S.) Next, M commences the Beyer/Levialdi shrinking algorithm. The shrinking step is modified so that if a point in S is removed, the contents of that cell's A register are added to the A register of a neighbor point in S (which is not removed).

Readily, when the region reduces to a single point, this cell's A register contains the number of border points of S. This count can now be shifted rightwards to the uppermost, leftmost point of S using the technique described in Section 4.1.

5.3. Compactness and elongatedness. Various measures are used for quantifying the shape complexity of a region. The compactness of a region is usually measured by P^2/A, where P is perimeter and A is area. We have just shown how a log-space BCA can compute and store P and A at the uppermost, leftmost point of a region S in O(diameter of S) time steps. If we assume multiplication and division take unit time, then two more time steps are required to complete the computation.

The elongatedness of a region is measured by A/W^2, where W is the number of "shrinking" steps required to annihilate the region. A shrinking step consists of the deletion of all border points of the region. Thus W is the maximum value in the distance transformation of the region. A log-space BCA can compute W and store it at the uppermost, leftmost point of a region S as follows. First, the designated cell is located and the minimum spanning tree rooted at that cell is constructed for cells in S. Simultaneously, the distance transformation of S is computed, each cell in S storing its distance from $\bar{S}$ in its D register. Next, the leaf cells initiate reply signals which move back to the designated cell. When a cell receives the reply signal it compares the contents of its D register with the contents of all its sons' D registers, and copies the largest value into its own D register. Thus after a cell receives the reply signal, its D register contains the maximum distance of any cell in its subtree from $\bar{S}$. In particular, the designated cell receives the reply signal

after O(intrinsic diameter of S) time steps. Again, if we assume multiplication and division are unit time operations, then elongatedness can be computed in two more time steps. Alternatively, we could again use the Beyer/Levialdi shrinking algorithm to find the maximum value in time proportional to the diameter of S.

5.4 Diameter

The diameter of a region S with respect to a point c is defined as the maximum distance between c and any point of S. A log-space BCA M can compute the 4-diameter of a region S with respect to a point c of S by using the minimum spanning tree of the picture rooted at c. At time step 1 cell c initializes two registers, A and B, to 0. Beginning at time step 2 c's A register starts counting at half speed and c initiates the construction of its minimum spanning tree. If when a cell becomes part of the tree it determines that it is in S but all of its sons are in $\overline{S}$, then the cell is a local maximum distance from c. Such a cell initiates a signal which propagates back to the root at unit speed. If two signals arrive simultaneously at a node, then they both must have originated at an equal distance from this cell, so only a single signal continues. Since the tree grows at unit speed and a signal returns at unit speed, when c receives the signal its A register contains the distance to the cell that originated the signal. Thus whenever c receives a signal, it copies the contents of its A register into its B register.

One of the four corners in the picture must be a maximum distance from c. Therefore when each of these corner cells becomes part of the tree, it initiates a nondestructable signal back to c. By the time c receives all four corner signals, no later than twice diameter time, it must have received all of the signals sent by the candidate farthest points of S.

So the number stored in c's B register must be the 4-diameter of S with respect to c.

Computing the diameter of S with a log-space BCA appears more difficult because running the above minimum spanning tree algorithm simultaneously from every point of S would require each cell to store O(area of S) arcs, one for each tree rooted at a point of S. We now show that an alternative method can be used to obtain an O(diameter of S) time solution to this problem. First, we derive some properties of the chessboard and city-block metrics which will be exploited to yield fast algorithms.

Theorem 5.1. The 8-diameter of a region is equal to the length of the longest side of the region's upright framing rectangle.

Proof: It is easily seen by the definition of chessboard distance that the maximum distance between any two points in an m by n upright rectangle is equal to max(m-1,n-1), i.e., the distance from the upper-left to the lower-right corner. In fact, this is the distance between any pair of points which are on opposite short sides of the rectangle. To see this, consider a rectangle m rows high and n columns wide, where $m \geq n$. The distance from an arbitrary point u in the top row to an arbitrary point v in the bottom row is equal to $d_8(u,v) = \max(m-1,|j-1|)$, where u is at coordinate (1,i) and v is at (m,j). For all $1 \leq i, j \leq n$, $|j-1| < n \leq m$, hence $d_8(u,v)=m-1$.

Given a region S, its upright framing rectangle S' just contains S so there must be points of S in the top and bottom rows and the left and right columns of S'. Since $S \subseteq S'$, the 8-diameter of S must be no greater than the 8-diameter of S'. But the existence of points of S on each side of S' implies that the 8-diameter of S equals the 8-diameter of S', which is just the length of the longest side of S'. //

In Appendix I we show that an arbitrary connected region's upright framing rectangle can be constructed by a BCA in O(diameter of S) time steps by repeatedly filling concave corners. That algorithm, while it is guaranteed to halt with the desired enclosing rectangle, does not specify how a BCA can detect its completion. Of course, since the framing rectangle of a region can be no larger than the picture itself, the BCA could easily be made to wait picture-diameter time steps in order to assure completion of the algorithm. This is not desirable, however, since in general S's diameter may be much smaller than the picture's diameter.

We now show how a log-space BCA M can check the completion of the propagation process in 2(h+w-2) time steps, where h and w are the height and width of the framing rectangle. Each time a cell becomes an upper-left corner, the cell starts a signal which travels clockwise around the region's border verifying whether or not the region currently constructed is an upright rectangle. If it is not, the signal dies. Contained in the

signal are the coordinates of the starting cell, so that the signal can recognize when the traversal is completed. If the starting cell is still an upper-left corner when its signal returns, this cell must be the true upper-left corner of the framing rectangle since the propagation process at each border point had stopped before the signal passed it.

In conjunction with this procedure, the signal can also maintain two counters, one which counts the number of rightward moves and one which counts the number of downward moves. Thus when the framing rectangle has been verified, the upper-left corner cell also contains the number of rows and columns in the rectangle. The maximum of these two numbers can then be computed by this cell in no more than log diameter time steps. Thus a log-space BCA can compute the 8-diameter of a region in a number of time steps proportional to the 8-diameter of the region.

We now consider the computation of the 4-diameter of a region.

Theorem 5.2. The 4-diameter of a region is equal to the length of the longest side of the region's tilted framing rectangle.

Proof: A region's tilted framing rectangle is the smallest enclosing rectangle with sides inclined at $\pm$ 45° to the picture's sides. The proof is analogous to the proof of Theorem 5.1, since again it can be shown, this time from the definition of

city-block distance, that any pair of points on opposite short sides of any tilted rectangle are at the same distance from one another. To see this, consider Figure 5.1 in which, without loss of generality, the short sides have slope -1. Let (x_1,y_1) be an arbitrary point on the side having y-intercept a and (x_2,y_2) an arbitrary point on the opposite side having y-intercept b. Since (x_1,y_1) and (x_2,y_2) are on the short sides of the tilted rectangle, it is readily seen that $x_2>x_1$ and $y_2>y_1$. This implies that $d_4((x_1,y_1),(x_2,y_2)) = (x_2-x_1)+(y_2-y_1) = (x_2+y_2)-(x_1+y_1)$. But the slopes of these sides imply that $x_1+y_1=a$ and $x_2+y_2=b$. Hence, $d_4((x_1,y_1),(x_2,y_2)) = b-a$, which is just the length of the longest side of the tilted rectangle. The theorem now immediately follows from the fact that a region's tilted framing rectangle by definition contains points of the region on each of its four sides. //

In analogy with the computation of a region's upright framing rectangle, it can be shown that a BCA can construct an arbitrary connected component S's tilted framing rectangle in O(diameter of S) time. Again, the completion of this construction can be detected by a log-space BCA in which each top corner cell sends a signal around the border checking whether or not the region currently constructed is a tilted rectangle and simultaneously counting the length of its sides. Once the true top corner cell has been found, it computes the maximum

of the two side lengths in at most log diameter more time steps.

Computing the intrinsic 4-diameter of a connected component S with respect to a given point c of S on a log-space BCA once again involves constructing a spanning tree. At time step 1 c initiates construction of the minimum spanning tree of cells in S rooted at c. Each leaf node is a local maximum distance from c, so each initiates a reply message which contains the distance of the longest path from the cell holding the signal to any of the leaf nodes in its subtree. That is, each leaf cell initializes its D register to 1. Each nonleaf cell in the tree waits until all of its sons contain the reply signal before receiving the signal and setting its D register to one plus the maximum of its sons' D register contents. Hence when c receives the reply signal after twice intrinsic 4-diameter of S time steps, its D register contains the intrinsic 4-diameter of S with respect to c.

Computing the intrinsic diameter of S is more difficult. If S is convex (see Section 5.6) then its intrinsic diameter is equal to its extrinsic diameter since the path between each pair of points of S is entirely contained in S. Therefore, in this case we can use the algorithms described above to compute the intrinsic diameter of S. Otherwise, intrinsic diameter need not be realized by a pair of points which touch the region's enclosing rectangle (see, e.g., Figure 5.2) or even

a pair of border points (see, e.g., Figure 5.3).

The intrinsic 4-diameter of a region S can be computed by the following sequential algorithm. First, find the uppermost, leftmost point c and construct the spanning tree of S rooted at this point. This requires O(intrinsic diameter of S) time. Since a cell has only four neighbors, each node in the tree has no more than three sons (one neighbor must be its father node). The root node c has no father, but since it is an upper-left corner in S, it has at most two sons in S. Therefore we can define a log-space BCA M which simulates an inorder traversal of the nodes of this spanning tree, staying long enough at each node to compute the intrinsic 4-diameter of S with respect to the current node. This involves constructing an overlying spanning tree rooted at the current node which does not conflict with the spanning tree rooted at c. Each node's "rooted" intrinsic 4-diameter requires O(intrinsic diameter of S) time steps to compute, hence the complete traversal of points of S takes O(intrinsic diameter^2) time. At the same time M also retains the maximum intrinsic diameter from any node visited so far, so that when the traversal is completed, the intrinsic 4-diameter of S will be stored at c.

5.5. Height and width

The height and width of a region S are the distances between the highest and lowest rows, and the leftmost and rightmost columns, of the picture that contain S, respectively. Since these are the dimensions of S's upright framing rectangle, we can compute them by first constructing S's rectangle and then counting the number of points on each side. In Section 5.4 we showed this procedure required O(diameter of S) time on a log-space BCA.

5.6 Convexity

A region S is called convex if every straight line that intersects S, intersects it in exactly one run of points of S. In digital pictures this definition requires some modification since a straight line will not in general pass through digital points. Sklansky et al. [28] and Smith [6] have used the notion of a minimum perimeter polygon to recognize convexity, but this approach does not detect shallow concavities.

Another definition of convexity uses the notion of a line of support. A line of support of a subset of points S through a point p of S is a line through p such that S lies entirely in one of the closed half planes bounded by this line. Then, a subset S is called convex if there exists a line of support through every border point of S. If S is a region in a digital picture, then we must modify this definition to allow for the discreteness of the data. Therefore, define a digital line of support of a region S through a point p of S to be a (real) line through p such that every border point of S is in or near one of the closed half planes bounded by this line. A digital point (i,j) is near the real point (x,y) if $\max(|x-i|, |y-j|)<1$.

We now describe how a log-space BCA M can decide whether or not a region S is convex in the above sense, in $O(\text{perimeter}^2)$

time after a preprocessing phase which takes O(diameter) time. M can check in O(diameter of S) time whether or not S is simply-connected using the Beyer/Levialdi algorithm (see Section 4.1), a necessary condition for S to be convex. Therefore we will assume that this process occurs in conjunction with, but does not interfere with, the action of M described below which assumes S is simply-connected. If S is not simply-connected, then M sends a reject signal to the uppermost, leftmost point of S.

First, M computes each cell's coordinates, finds the leftmost, uppermost point of the outer border of S, and then resynchronizes the array in O(diameter) time as described in Section 4.1. During the next two time steps each border point marks itself and determines its successor and predecessor border points. Next, each border point stores two copies of its coordinates in two separate "channels." Beginning at the next time step the cyclic ordered list of coordinates in channel 1 shift clockwise around the border at unit speed. That is, at each time step each border point copies the coordinates stored in the first channel of its predecessor. Assuming that the distinguished cell specially marks the copies of its coordinates, this cell can detect when its coordinates pass by every O(perimeter) time steps. At such times the coordinates in the second channel shift one position counterclockwise around the border. That is, each border point copies the coordinates stored in the second channel of its successor.

In this way the coordinates of S's border points shift clockwise around the border at unit speed and simultaneously counterclockwise at 1/perimeter speed. Thus at each time step every point on the border stores the coordinates of three border points, one point p in channel 1, one point q in channel 2, and itself r. At each time step, each border point now determines the interior (i.e., counterclockwise) angle between $\vec{rp}$ and $\vec{rq}$, as shown in Figure 5.4. If this angle is less than or equal to 180° or the distance from r to the line segment $\overline{pq}$ is less than one, then p and q lie on the inside of the tangent line through r. After O(perimeter^2) time every triple of border points has been checked. At this time (when the distinguished cell's three points are all the same) the distinguished cell sends a signal around the border gathering the results of the perimeter^3 tests. If no concavities were detected by any border point, then the distinguished cell enters an accepting state. Thus this is an O(perimeter^2) algorithm for detecting convexity.

6. Memory-augmented cellular pyramids

Cellular pyramids, introduced in [29,30], can accept many languages in time proportional to the logarithm of the diameter of the input array. In this section we consider memory-augmented cellular pyramids, showing that for a moderate increase in memory we significantly enhance their capabilities.

6.1 Definitions

A pyramid cellular acceptor (PCA) is a pyramidal stack of bounded cellular automata, where the bottom layer has dimensions 2^n by 2^n, the next lowest 2^{n-1} by 2^{n-1}, and so on, until the (n+1)st layer consists of a single cell. Each cell now has nine neighbors--four son cells in a 2 by 2 block in the level below, its four horizontal and vertical neighbors, called brothers, in its own level, and one father cell in the level above. That is, if a cell's coordinates in its level are (i,j), then its son cells are at coordinates (2i,2j), (2i-1,2j), (2i,2j-1), and (2i-1,2j-1) in the level below; its brothers are at coordinates (i-1,j), (i+1,j), (i,j-1), and (i,j+1) in the current level; and its father's coordinates are ($\lceil i/2 \rceil$, $\lceil j/2 \rceil$) in the level above. Thus the transition function of a cell in a PCA maps 10-tuples of states into sets of states in the nondeterministic case, or into states in the deterministic case. A 2^n by 2^n input array over the input state set defines the initial states of the cells in the bottom layer of a PCA;

all other cells are initialized to a quiescent state. Again, we surround the entire pyramid by a border of cells permanently in the boundary state in order to restrict a computation to a fixed set of cells. The single cell in the (n+1)st layer is called the root and is the accepting cell.

Alternative definitions can be made which restrict the neighbors of a cell in a PCA. In particular, if each cell has only its four sons as neighbors, then information can only move up the pyramid. Such a variant PCA will be called a bottom-up pyramid acceptor (UPCA).

In either case, we can augment each cell of a PCA or UPCA with more memory in the same way that it was added to BCA's. Thus an $L(2^n)$-space PCA is a PCA in which each cell has a two-way read-write storage tape, no more than $L(2^n)$ squares of which are scanned during a computation on an array of size 2^n by 2^n. If $L(2^n)=2n$, then we call the acceptor a log-space PCA since each cell has storage equal to the logarithm of the input array's area. Notice that the additional hardware cost of a log-space PCA over a BCA is moderate--with less than a third more cells and a total memory size of $O(n \cdot 2^{2n})$ instead of $O(2^{2n})$.

Adding memory to a UPCA should be done nonuniformly since cells in the bottom layers only depend on a few cells in their bases, while cells in the top layers depend on almost all of the input array. Therefore, we define an L(k)-space UPCA

to be a UPCA in which the amount of storage each cell is allowed is a function of its level k in the acceptor. If $L(k)=2^{2k}$, then each cell has storage size equal to its base's area. The total memory requirements for a UPCA with input size 2^n by 2^n is $1\cdot 2^{2n}+4\cdot 2^{2n-2}+\ldots+2^{2k}2^{2n-2k}+\ldots+2^{2n}=(n+1)2^{2n}$, i.e., O(area log area). If $L(k)=2^{k+1}$, then each cell has storage size equal to its base's diameter. In this case the total memory requirements are only $2\cdot 2^{2n}+4\cdot 2^{2n-2}+\ldots+2^{k+1}\cdot 2^{2n-2k}+\ldots+2^{n+1}=2^{2n+2}$, i.e., only four times the area of the input array! Restricting the memory at each cell even further, to $L(k)=2k$, means each cell has storage size equal to the logarithm of its base's area. The total memory requirements here are $2^{2n}+2\cdot 2^{2n-2}+\ldots+2k2^{2n-2k}+\ldots+2n$, i.e., less than twice the area of the input array, or less than twice the memory used by a conventional BCA (assuming unit storage per cell in a BCA). When $L(k)=2k$ we call the acceptor a <u>log-space UPCA</u>.

6.2 Log-space PCA's

Augmenting each cell in a PCA with an amount of memory equal to the log of the area of the input array simplifies many PCA algorithms as well as making more tasks possible to compute. In particular, a log-space PCA can naturally store reduced resolution versions of a given picture, computed by repeated 2 by 2 averaging, so that a pyramidal stack of coarse to fine images can be hierarchically processed. See [31-33] for applications of this pyramidal data structure. All of the algorithms described in Sections 3-5 for log-space BCA's are now applicable on a set of specified levels in log-space PCA's. For example, coarse-fine template matching [34] is very appropriate here--a coarse template is matched against a coarse resolution version of the picture, and then for each point where a match appears promising, a signal is sent down the PCA to the corresponding point in a finer resolution copy; each such point then applies a finer resolution template at that point.

More generally, all of the region and picture representations discussed in Section 4 can be computed at various levels of coarseness. Subsequent processes can then perform top-down search operations based on these approximations.

The quadtree representation is ideally suited to a log-space PCA since each node in the tree is in one-to-one

correspondence with the cells in the PCA, and neighbors in the tree are also neighbors in the PCA. Thus this representation can now be computed from its array representation in O(log diameter) time. For the same reasons, search operations using a quadtree stored in a log-space PCA require O(log diameter) time. See [35] for a discussion of efficient operations on quadtrees.

Log-space PCA's can also perform other tasks, such as local property counting, but we defer their presentation to the next section since they can also be computed by log-space UPCA's.

6.3 Memory-augmented UPCA's

In the case of log-space UPCA's, each cell has O(log base-area) storage, enough to count picture properties or store pixel coordinates, for example. It was shown in [29] that a one-dimensional UPCA can count local properties in O(log diameter) time, but that algorithm could not be generalized to two dimensions. An O(diameter) time algorithm was given, however, for detecting two-dimensional local properties by implementing at each cell a sequential scan of the bounded width slabs of cells in its base which are centered on the cracks between its sons' bases. Figure 6.1 shows the relevant cells in the horizontal and vertical slabs as shaded.

On a log-space UPCA, extending this algorithm to count occurrences of a local property is straightforward. That is, each cell c is equipped with a counter initialized to zero. After c's sons have computed their counts, c sums them in its counter while simultaneously scanning its slabs for occurrences of the property which overlap its sons' bases. For each such occurrence c increments its counter. If c is in level k, then this process of determining c's count given its sons' counts takes $O(2^k)$ time using either a unit cost or "bitwise" cost criterion. Letting T(n) be the time to count local properties at a cell in level n, we get the following recurrence relation using this algorithm

$$T(n) = \begin{cases} a & \text{, for } n=0 \\ T(n-1)+a\cdot 2^n & \text{, for } n>0 \end{cases}$$

for constant a. It is immediate that $T(n) = O(2^n)$, i.e., a log-space UPCA can count local properties in O(diameter) time.

In the particular case of counting point properties, e.g., for computing the area of a region or the histogram of a picture, there is no chance of boundary overlap. Thus each cell only has to sum its four sons' counts before passing its count up to its father. If we use the unit cost criterion for addition, then counting point properties takes O(log diameter) time; using the logarithmic cost criterion implies counting point properties takes $O(\log^2 \text{diameter})$ time.

Non-local properties such as height and width of a region S can also be computed in $O(\log^2 \text{diameter})$ time. Specifically, let each cell in a log-space UPCA M have five registers L, R, U, D, and B. The first four of these will store at each cell c the matrix coordinates (relative to c's base) of the leftmost and rightmost columns, and the uppermost and lowermost rows of that part of S which intersects c's base, respectively. The B register is a Boolean variable used to indicate whether or not S intersects c's base at all. A cell c computes its register values from its sons' values as follows. For each son that intersects S, c recomputes its coordinates to be relative to c's base. c's lowest row's coordinate is the minimum of those sons' L register contents. The other three coordinates are computed analogously. Finally, c computes R-L+1 and D-U+1, which are the width and height of S in c's base. The recurrence

that results is

$$T(n) = \begin{cases} a & , n=0 \\ T(n-1) + a \cdot F(n) & , n>0 \end{cases}$$

where a is a constant and F(n) is the time required for addition and maximum operations on n-bit quantities. Using the unit cost criterion implies F(n)=1, so T(n)=O(log diameter) time; with the logarithmic cost criterion F(n)=n, resulting in $T(n)=O(\log^2 \text{ diameter})$ time.

The method used in finding height and width can be more generally thought of as the use of the well-known divide-and-conquer technique for recursive search or property measurement. Here, we decompose a problem on a 2^n by 2^n array into four problems on 2^{n-1} by 2^{n-1} arrays plus a single "sewing" problem which mends the partial results together. Other problems of this form include finding the parity or predominant gray level of a binary picture, computing the average or maximum gray level of a region, and finding the leftmost point in the uppermost row of a region. This last problem implies that a distinguished cell in a region can be marked by a log-space PCA in $O(\log^2 \text{ diameter})$ time using the logarithmic cost criterion. This problem was seen in Sections 4 and 5 to be a basic operation in other problems; hence its efficient solution is important for many image analysis tasks.

Recently, many problems in computational geometry have been solved using divide-and-conquer over Euclidean k-space [36-38]. For example, Bentley and Shamos [36] give an algorithm for finding the closest pair of points in 2-space by

recursively halving the plane, finding the closest pair in each half-plane, and then "patching up" the tentative solution by examining points near the dividing line. We now describe how a log-space UPCA can find the closest pair (using city-block distance) of 1's in a binary array by modifying their technique and the slab method mentioned above. Assume a cell c in level k has received the distance between the closest pair of 1's in each of its sons' bases. Let δ denote the minimum of these four numbers, computed by c in constant time by the unit cost criterion. To obtain the minimum distance between all pairs of points in c's base, it suffices to examine only those points within δ of a border of a son's quadrant, in order to check if there exists a pair in different quadrants which are closer than δ apart. Consider, for example, the border between c's upper-left son's quadrant and its upper-right son's quadrant (Figure 6.2). The Bentley algorithm uses a running window of size 2δ by 2δ centered on this border line, and computes distances between all possible pairs of points within this window. Unfortunately, the slab method used for local property counting cannot be generalized to a variable slab width. However, as noted by Bentley, since the minimum separation between pairs of points in either 2δ by δ half-slab window is δ, there can only be a constant number of points in the window. Figure 6.3 shows the worst case when using city-block distance, i.e., eight points in each half-slab. Clearly, only the point closest to the border

in each row needs to be considered, so we can reduce the maximum number of points for consideration in each half-slab window to five, since at most five rows can be nonzero when using the city-block metric.

Now suppose each of c's sons sequentially sends to c the column coordinates of the leftmost and rightmost 1's in each row, while simultaneously sending the row coordinates of the uppermost and lowermost 1's in each column. Analgously to the method described in [30], c can compute the coordinates of its extrema and send them to its father. At the same time, c tests whether each coordinate is within δ of the border. If it is, it is stored in the fifth of five registers, the contents of registers 2-5 being simultaneously shifted up into registers 1-4. Thus c has forty registers, five for each half-slab window scanning the four touching borders between c's sons' bases. Each set of five registers keeps track of the five most recent points which are candidates for being paired with another point in the adjacent half-slab. δ time steps after a point was stored in register 5, it is in the center of its half-slab and all points which it could possibly be paired with are currently stored in the adjacent half-slab. Thus c computes the distance from this point to each of the other five points and determines whether or not there is a pair closer than δ apart. If we use the unit cost criterion for all of the operations involved, then c takes O(diameter) time to determine the distance between its closest pair of 1's.

Thus the recurrence relation describing the algorithm is $T(n) = T(n-1) + O(2^n)$, giving an O(diameter) time algorithm. (Using the logarithmic cost criterion implies an O(diameter log diameter) algorithm.)

2^{k+1}-space UPCA's have enough storage so that a cell can store a description of the border of its base. This additional memory simplifies many of the algorithms described for log-space UPCA's. However, it doesn't speed them up since a cell still has to sequentially scan its border, no matter whether it is sent by its sons or stored internally.

2^{2k}-space UPCA's have O(base-area) storage per cell. In particular, the root of a 2^{2k}-space UPCA can store the entire input array. Hence, a 2^{2k}-space UPCA can easily simulate a BCA.

7. One-way log-space parallel/sequential acceptors

This section establishes the advantages of additional memory in a class of acceptors of rectangular input arrays, called parallel/sequential acceptors [39-40], which is a compromise between sequential and cellular automata. Informally, a parallel/sequential acceptor is a one-dimensional cellular acceptor that reads one row of its rectangular input array at a time, and moves up and down as a fixed unit to scan the array. The class of languages accepted by parallel/sequential acceptors is equivalent to the class of bounded cellular array languages. However, if we restrict the movement of the acceptor so as not to allow it to move upward, then it is known [40] that this "one-way" parallel/sequential acceptor is strictly weaker than the two-way acceptor. This section establishes the increased power of one-way parallel/sequential acceptors when each cell is augmented with an amount of storage proportional to the logarithm of the size of the array.

7.1 Definitions

A parallel/sequential acceptor (PSA) is a 7-tuple $M=(Q,\Sigma,\delta,\mu, q_o,\#,Q_A)$, where Q is a finite, nonempty set of states, Σ is a finite, nonempty set of tape symbols, $\delta:Q^3x\Sigma\rightarrow Qx\Sigma$ is the state transition function if M is deterministic, $\delta:Q^3x\Sigma\rightarrow 2^{Qx\Sigma}$ if M is nondeterministic, $\mu:Qx\Sigma\rightarrow\{-1,0,1\}$ is the move function, $q_o\in Q$ is the initial state, $\#\in Q$ is the boundary state, and $Q_A\subseteq Q$ is the set of accepting states. Given an input array of size m by n, M consists of a string of cells $c_1,\ldots,c_n$, one located at each column of the input. The next state of any cell c_i depends on the current states of cells c_{i-1}, c_i, and c_{i+1} and the current symbol at c_i's position. At each step c_i can write a new symbol at its position depending on the same tuple of states and single symbol. Cells c_1 and c_n are defined specially so that their undefined neighbor cells are regarded as permanently in the boundary state #. The move function is defined only at cell c_1, and specifies the motion of M at the end of each step, i.e., whether M moves up or down one row, or does not move at all. A step of computation consists of the simultaneous application of the transition function at each cell, and the subsequent repositioning of M at an adjacent row of the array. If M moves up off the top row or down off the bottom row, then we require M to move back onto the array at the next step. (We can do this by adding 0th and (n+1)st rows containing

special nonrewritable boundary symbols which M's move function can detect.) A configuration of M is a triple (T,α,i), where T is the m by n array of symbols currently written on the tape, α is an n-tuple of states specifying the current states of the cells in M, and i is an integer between 1 and m giving the row on which M is currently positioned. An m by n array T_o is accepted by PSA M if, given the initial configuration $(T_o,(q_o,\ldots,q_o),1)$, a sequence of steps causes c_1 to enter a state in Q_A. The language of M is defined to be the set of all arrays accepted by M.

A one-way parallel/sequential acceptor (OPSA) is a PSA in which the range of the move function is restricted to {0,1}, i.e., the acceptor is not allowed to move upward. Obviously, the ability to write provides no advantage for OPSA's, so in this case a configuration is defined as a pair (α,i).

An L(m,n)-space OPSA is an OPSA in which each cell contains a finite control, a read-only input head, and a two-way read-write storage tape, of which no more than L(m,n) squares are used during a computation on an array of size m by n. If L(m,n)=log mn, then we call the acceptor a log-space OPSA. The necessary changes in the transition function to convert an OPSA to an L(m,n)-space OPSA are analogous to those described in Section 2.1 for an L(n)-space BCA. Notice that this definition of L(m,n)-space OPSA is a two-dimensional version of an on-line tape-bounded Turing acceptor with the conditions that all read heads move in lock step and the finite controls can sense their nearest neighbors' states.

7.2 Comparison with other types of acceptors

In this section we prove that OPSA's are strictly weaker than log-space OPSA's by showing that OPSA's cannot accept all of the two-dimensional rectangular finite-state languages, whereas log-space OPSA's can accept this class. All acceptors will be assumed to be deterministic. First, we show

Theorem 7.1. The class of languages accepted by OPSA's is incomparable with the class of rectangular array languages accepted by two-dimensional finite-state acceptors.

Proof: On a 1 by n array, an OPSA can simulate a linear bounded automaton, which can accept non-finite-state languages such as $\{a^n b^n | n \geq 1\}$. Conversely, consider the set L of rectangular arrays over the alphabet {0,1,2} such that each 1 has either zero, one, or two 1's and 2's as 4-neighbors, each 2 has four 1's as 4-neighbors, and there is an arc from the lower-left to the lower-right corner. An arc is a 4-path of 1's and 2's such that the predecessor and successor of each 2 in the sequence are either both horizontal or both vertical neighbors of the 2. By the conditions on the neighbors of 1's and 2's in L it is easily seen that an arc in L defines a deterministic path of cells, given a starting cell and its neighbor, since after moving onto a 1 there is at most one unvisited neighbor, and 2's must be crossed deterministically. Thus the 1's form a set of disjoint paths and the 2's occur only where these paths cross, acting as "bridges". In particular, this implies that a deterministic finite-state acceptor

can accept L by moving to the lower-left corner and following the arc that begins there, accepting if it ever reaches the lower-right corner.

We now show that no OPSA can accept L. Consider the set of (2n) by (4n+1) arrays of 0's, 1's, and 2's in which the bottom row is 0101...010, and the array contains a set of n inverted u-shaped arcs connecting distinct pairs of 1's in the bottom row. For example, representing 0's by blanks, the following array connects the first 1 in the bottom row with the third, the second with the sixth, the fourth with the eighth, and the fifth with the seventh.

```
    111111111
    1       1
  1111211111  1
  1   1   1   1
 11211 1 11211 1
 1 1 1 1 1 1 1 1
```

It is easily verified that 2n rows are sufficient to construct all of these arcs without touching. The number of possible pairings of 1's in the bottom row is $\binom{2n}{2}\binom{2n-2}{2}\cdots\binom{2}{2}=(2n)!/2^n$; hence there are $(2n)!/2^n$ such arrays, A_i, that are topologically distinct.

Suppose, in contradiction, that there is an OPSA M with state set Q that accepts L. For any two arrays A_i and A_j, we can easily construct a 2 by (4n+1) array B_{ij} such that the resultant (2n+2) by (4n+1) array obtained by concatenating B_{ij} to the bottom of A_i is in L, but the array obtained by concatenating

B_{ij} to the bottom of A_j is not in L. That is, since A_i and A_j are topologically distinct, there exists a pair of columns h and k, where $h<k$, whose bottom elements are connected 1's in A_i but not in A_j. Therefore construct B_{ij} as follows: the first row is all 0's except for 1's in columns h and k, and the second row is a string of h 1's, followed by a string of (k-h-1) 0's, followed by a string of (4n-k+2) 1's. It follows that M's configurations after scanning A_i and A_j must be distinct since either could lead to acceptance depending on the contents of the remaining rows. But there are only $|Q|^{4n+1}$ possible configurations of M after scanning the (2n)th row, and this number is less than $(2n)!/2^n$. Thus the assumption that M accepts L must be false. //

Theorem 7.2. A log-space OPSA can simulate a two-dimensional finite-state acceptor.

Proof: Given a finite-state acceptor A with state set $Q=\{q_1,q_2,\ldots,q_k\}$ and transition function δ, construct a log-space OPSA M as follows. Let the input tape be m by n. Each cell c_i of M will construct a length k vector V_i of (column number, state, acceptance logical variable) triples. These vectors will be updated each time M moves downward so as to maintain the following interpretation of their contents. M at row α and $V_\beta(i)=(\gamma,j,t)$ implies that if A is ever at position (α,β) in state q_i, then A will eventually move to row $\alpha+1$ for the first time in column γ and in state q_j. If t=1 then A entered an accepting state during this sequence of

moves. If j=0 then A will never enter row $\alpha+1$ starting from the given configuration. Clearly there is sufficient storage in M to store these vectors; details will not be given here. In addition, M marks the unique entry in one of these vectors which indicates where A first moves below the current row when started in its initial state at the upper-left corner. Hence if at some time the marked entry's acceptance variable is 1, then that cell propagates an acceptance signal to cell c_1, and M accepts.

We now sketch how M can compute these V_i vectors. At row 1 each cell c_β reading symbol x determines for each state in Q where A will first move off this row. This is done for each possible configuration of A in row 1 by chaining through left and right moves of A until either A moves up, down, or off of the left or right ends of the row. Formally, for each state $q_i \in Q$ define

$$V_\beta(i) = \begin{cases} (\beta,j,t), & \text{if } \delta(q_i,x)=(q_j,\text{down}) \\ (0,-,t), & \text{if } \delta(q_i,x)=(q_j,\text{up}) \text{ or} \\ & \quad (\delta(q_i,x)=(q_j,\text{left}) \text{ and } \beta=1) \text{ or} \\ & \quad (\delta(q_i,x)=(q_j,\text{right}) \text{ and } \beta=n) \\ (\text{left},j,t), & \text{if } \delta(q_i,x)=(q_j,\text{left}) \\ (\text{right},j,t), & \text{if } \delta(q_i,x)=(q_j,\text{right}) \end{cases}$$

where t=1 if q_i is an accepting state in A, 0 otherwise.

At subsequent time steps until V_β is completely defined, c_β copies its neighbors' vectors, $V_{\beta-1}$ and $V_{\beta+1}$, and updates its own vector as follows: If $V_\beta(i)=(\text{left},j,t)$ and $V_{\beta-1}(j) =$

AD-A065 328 MARYLAND UNIV COLLEGE PARK COMPUTER SCIENCE CENTER F/G 5/8
MEMORY-AUGMENTED CELLULAR AUTOMATA FOR IMAGE ANALYSIS.(U)
NOV 78 C R DYER AFOSR-77-3271
UNCLASSIFIED CSC-TR-710 AFOSR-TR-79-0126 NL

2 OF 2
AD
A065328

END
DATE
FILMED
4 -79
DDC

(γ,h,s), then set $V_\beta(i)=(\gamma,h,s \vee t)$. Similarly, if $V_\beta(i)$ = (right,j,t) and $V_{\beta+1}(j) = (\gamma,h,s)$, then define $V_\beta(i)=(\gamma,h,s \vee t)$. Since these vectors have bounded length, we will assume that this copy operation takes unit time. Thus after at most kn time steps all of M's vectors will be properly defined.

Now assume that M has just moved down to row $\alpha>1$ and the V_β's are defined according to the induction hypothesis for row $\alpha-1$. As described for row 1, M determines, for each possible starting column and state, where A will first exit row α, storing this information in temporary vectors, W_β. If A moves off either the left or right end, i.e., $W_\beta(i)=(0,-,t)$, and $V_\beta(i)=(\gamma,j,s)$, then set $V_\beta(i)=(0,-,s \vee t)$. If A moves down to row $\alpha+1$, then define $V_\beta(i)=W_\beta(i)$. Otherwise, A moves up to row $\alpha-1$, say in state q_j, eventually returning to row α for the first time in column γ and state q_h, as specified in $V_\beta(j)$. For each such entry cell c_β must access $W_\beta(h)$, stored in cell c_β, in order to determine where A moves next, i.e., whether it moves down out of this row, or again moves up into the top $\alpha-1$ rows of the array. This access is accomplished by having the V_β's and W_β's simultaneously shift left and right, each cell picking off the necessary information as it moves past. This operation requires n time steps. This procedure is repeated until either A moves downward to row $\alpha+1$ or off the edge of the array. Since from a given starting state and column, A may oscillate between row α and the block

of α-1 rows above it at most kn times, M requires at most kn^2 time steps in order to define the V_β's at row α. Thus kmn^2 (i.e., O(diameter^3)) time is sufficient to simulate a two-dimensional finite-state acceptor by a log-space OPSA. (A finite-state acceptor itself takes O(diameter^2) time to accept any nontrivial finite-state language.) //

Corollary 7.1. The class of languages accepted by OPSA's is strictly contained in the class of languages accepted by log-space OPSA's.

Proof: Immediate from Theorems 7.1 and 7.2. //

7.3 Capabilities

Selkow shows [39] that his version of the OPSA can compute such properties as area, number of connected components, and number of occurrences of a given local property in height of the array time steps. His definition does not, however, restrict the cells to be identical, finite-state, or even have a bounded number of neighbors. In addition, acceptance is defined using a counter of unbounded size, "hardwired" to an unbounded number of cells, and which can sum in one step an unbounded number of inputs. In this section we show that log-space OPSA's, which are more conventionally defined, can measure a variety of geometrical properties which OPSA's cannot.

For example, point property counting is easily performed by a log-space OPSA M by having each cell increment a counter each time it scans the given property. At the bottom row, these counts are passed to the leftmost cell which sums them as they arrive. M moves down the picture at unit speed, and then spends array width plus log diameter time steps to obtain the final count. Thus the algoirthm takes O(diameter) time.

Counting local properties can be done similarly. Say we are counting a property of size k by ℓ. Then a log-space OPSA must spend $\ell/2$ time steps on each row so that each cell can gather the input symbols read at the current row by all cells within $\ell/2$ of it. Furthermore, each cell saves the most

recently read k such ℓ-tuples in order to reconstruct the most recently complete k by ℓ window centered at the cell. Each cell also has a counter as before, so the complete algorithm is still O(diameter) time. An OPSA cannot count local properties, on the other hand, since on a one-column array it is simply a finite-state automaton.

8. Conclusions and summary

In this paper we have investigated how augmenting bounded cellular, pyramid cellular, and parallel/sequential automata with memory proportional to the logarithm of the input array enhances the capabilities of these parallel models and simplifies algorithm design. The concept of memory-augmented cellular automata is an important one since it enables a more practical consideration of the effectiveness of these models for performing representative tasks. In particular, we have shown that log-space BCA's and log-space PCA's can efficiently perform a variety of tasks which are basic to image processing. Tables 8.1 and 8.2 summarize some of these results.

Task	Time
Region representation	
Region labeling	O(dia)
Run length coding	
construction	O(dia)
output	O(perimeter)
Chain coding	
construction	O(constant)
output	O(perimeter)
Distance transformation	O(dia)
Medial axis transformation	O(dia)
Quadtree construction	O(dia)
Picture description properties	
Histogram construction	O(dia)
Cooccurrence matrix construction	O(dia)
Centroid	O(dia)
Moments of inertia	O(dia)
Autocorrelation	$O(area^2)$
Region description properties	
Area	O(dia)
Perimeter	O(dia)
Compactness	O(dia)
Elongatedness	O(dia)
Diameter	O(dia)
Intrinsic diameter	$O(intrinsic\ dia^2)$
Height	O(dia)
Width	O(dia)
Convexity	$O(perimeter^2)$

Table 8.1. Some image analysis tasks and their computation times on log-space BCA's using the unit cost criterion for basic arithmetic and register transfer operations.

Task	Time
Quadtree construction	O(log dia)
Point property counting	O(log dia)
Local property counting	O(dia)
Height and width of a region	O(log dia)
Uppermost, leftmost point of a region	O(log dia)
Histogram construction	O(log dia)
Closest pair of points	O(dia)

Table 8.2. Some image analysis tasks and their computation times on log-space PCA's using the unit cost criterion for arithmetic and register transfer operations.

References

1. A.W. Burks, Ed., Essays on Cellular Automata, University of Illinois Press, Urbana, Ill., 1970.

2. S.N. Cole, Real-time computation by n-dimensional iterative arrays of finite-state machines, IEEE Trans. on Computers C-18, 1969, 349-365.

3. W.T. Beyer, Recognition of topological invariants by iterative arrays, MAC TR-66, M.I.T., Cambridge, MA., 1969.

4. F.C. Hennie III, Iterative Arrays of Logical Circuits, M.I.T. Press, Cambridge, MA., 1961.

5. A.R. Smith III, Cellular automata and formal languages, Proc. 11th IEEE Symp. on Switching and Automata Theory, 1970, 216-224.

6. A.R. Smith III, Two-dimensional formal languages and pattern recognition by cellular automata, Proc. 12th IEEE Symp. on Switching and Automata Theory, 1971, 144-152.

7. A.R. Smith III, Real-time language recognition by one-dimensional cellular automata, J. Computer Systems Sciences 6, 1972, 233-253.

8. S.R. Kosaraju, On some open problems in the theory of cellular automata, IEEE Trans. on Computers C-23, 1974, 561-565.

9. A. Rosenfeld and A.C. Kak, Digital Picture Processing, Academic Press, New York, 1976.

10. B.H. McCormick, The Illinois pattern recognition computer--ILLIAC III, IEEE Trans. on Computers C-12, 1963, 791-813.

11. M.J.B. Duff, CLIP 4: a large scale integrated circuit array parallel processor, IJCPR3, 1976, 728-733.

12. L. Fung, a high-speed image processing computer, Proc. 17th Annual ACM Technical Symposium, 1978, 11-17.

13. R.E. Stearns, J. Hartmanis, and P.M. Lewis II, Hierarchies of memory limited computations, Proc. 6th IEEE Symp. on Switching Circuit Theory and Logical Design, 1965, 179-190.

14. P.M. Lewis II, R.E. Stearns, and J. Hartmanis, Memory bounds for recognition of context-free and context-sensitive languages, Proc. 6th IEEE Symp. on Switching Circuit Theory and Logical Design, 1965, 191-202.

15. Y. Igarashi, Tape bounds for some subclasses of deterministic context-free languages, Info. Control 37, 1978, 321-333.

16. J. Hartmanis and L. Berman, A note on tape bounds for SLA language processing, Proc. 16th IEEE Symp. on Foundations of Computer Science, 1975, 65-70.

17. R.W. Ritchie and F.N. Springsteel, Language recognition by marking automata, Info. Control 20, 1972, 313-330.

18. K. Morita, Computational complexity in one- and two-dimensional tape automata, Doctoral thesis, Osaka University, Osaka, Japan, February 1978.

19. J. Hartmanis, On non-determinancy in simple computing devices, Acta Info. 1, 1972, 336-344.

20. I.H. Sudborough, On tape-bounded complexity classes and multi-head finite automata, J. Computer Systems Sciences 10, 1975, 177-192.

21. W.J. Savitch, Relationships between nondeterministic and deterministic tape complexities, J. Computer Systems Sciences 4, 1970, 177-192.

22. L. Cordella, M.J.B. Duff, and S. Levialdi, Thresholding: a challenge for parallel processing, Computer Graphics Image Processing 6, 1977, 207-220.

23. A. Rosenfeld, Iterative methods in image analysis, Proc. IEEE Conf. on Pattern Recognition and Image Processing, 1977, 14-18.

24. J.E. Hopcroft and J.D. Ullman, Nonerasing stack automata, J. Computer Systems Sciences 1, 1967, 166-186.

25. A. Rosenfeld, Picture Languages, Academic Press, New York, 1979.

26. S. Levialdi, On shrinking binary picture patterns, CACM 15, 1972, 7-10.

27. I. Shinahr, Two- and three-dimensional firing-squad synchronization problems, Info. Control 24, 1974, 163-180.

28. J. Sklansky, L.P. Cordella, and S. Levialdi, Parallel detection of concavities in cellular blobs, IEEE Trans. Computers C-25, 1976, 187-196.

29. C.R. Dyer and A. Rosenfeld, Cellular pyramids for image analysis, Technical Report 544, Computer Science Center, University of Maryland, College Park, MD., May 1977.

30. C.R. Dyer, Cellular pyramids for image analysis, 2, Technical Report 596, Computer Science Center, University of Maryland, College Park, MD., November 1977.

31. A. Hanson and E. Riseman, Preprocessing cones: a computational structure for scene analysis, COINS Technical Report 75C-7, Department of Computer and Information Science, University of Massachusetts, Amherst, MA., 1974.

32. S. Tanimoto and T. Pavlidis, A hierarchical data structure for picture processing, Computer Graphics Image Processing 4, 1975, 104-119.

33. B.K.P. Horn and B.L. Bachman, Using synthetic images to register real images with surface models, CACM 21, 1978, 914-924.

34. A. Rosenfeld and G.J. VanderBrug, Coarse-fine template matching, IEEE Trans. Systems, Man, Cybernetics 7, 1977, 104-107.

35. G.M. Hunter and K. Steiglitz, Operations on images using quad trees, IEEE Workshop on Pattern Recognition and Artificial Intelligence, June 1977, 25-28.

36. J.L. Bentley and M.I. Shamos, Divide-and-conquer in multidimensional space, Proc. 8th ACM Symp. on Theory of Computing, 1976, 220-230.

37. M.I. Shamos and D. Hoey, Closest-point problems, Proc. 16th IEEE Symp. on Foundations of Computer Science, 1975, 151-162.

38. M.I. Shamos and D. Hoey, Geometric intersection problems, Proc. 17th IEEE Symp. on Foundations of Computer Science, 1976, 208-215.

39. S.M. Selkow, One-pass complexity of digital picture properties, J.ACM 19, 1972, 283-295.

40. A. Rosenfeld and D.L. Milgram, Parallel/sequential array automata, Info. Proc. Letters 2, 1973, 43-46.

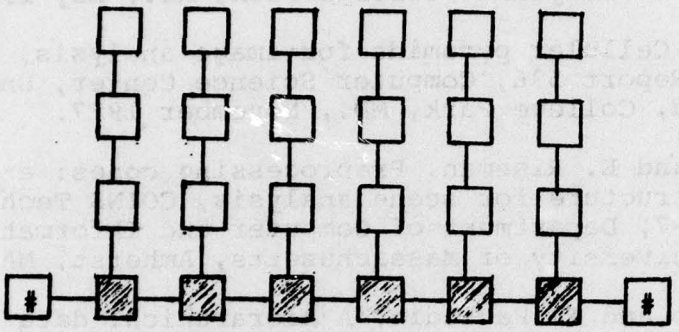


Figure 3.1

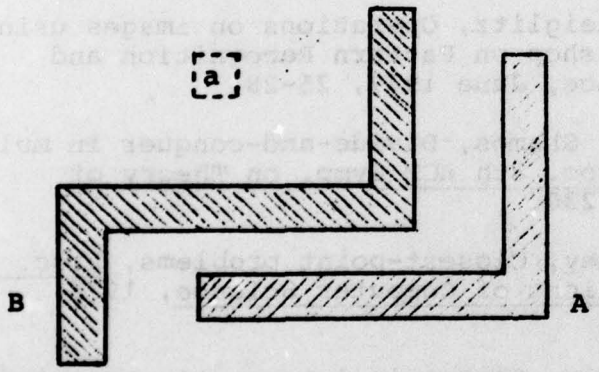


Figure 4.1

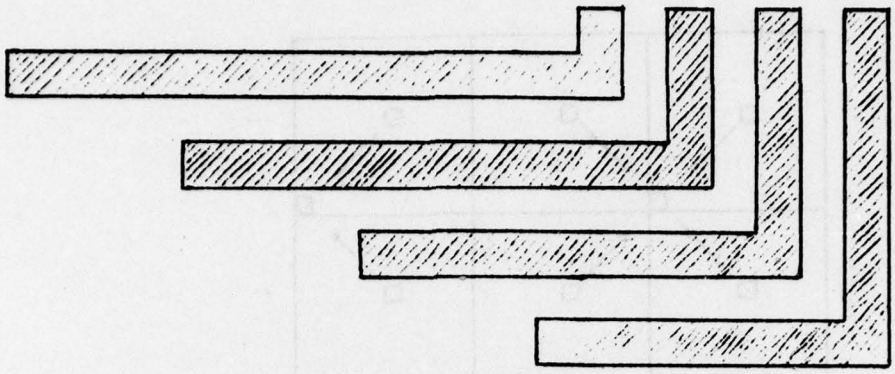


Figure 4.2

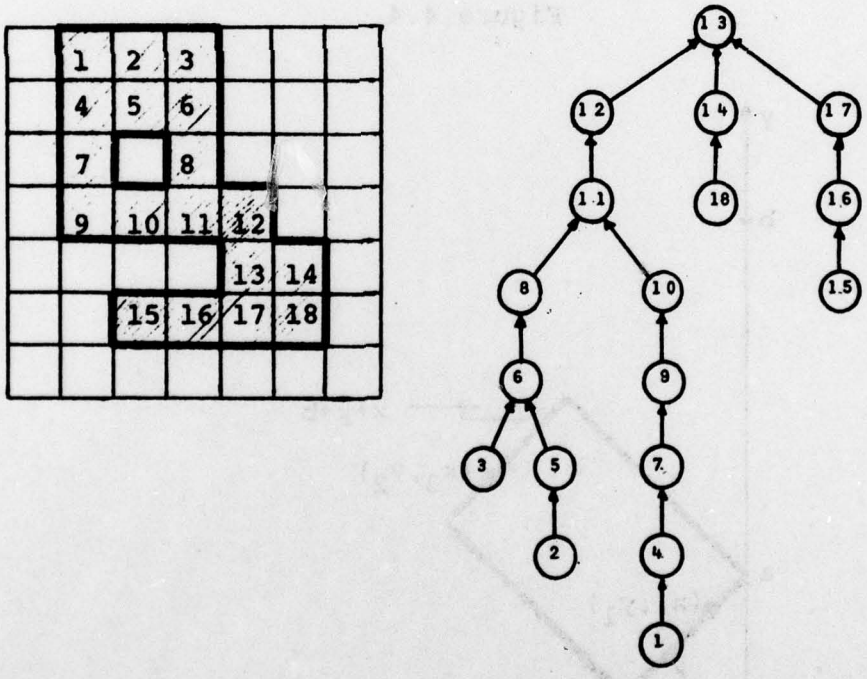


Figure 4.3

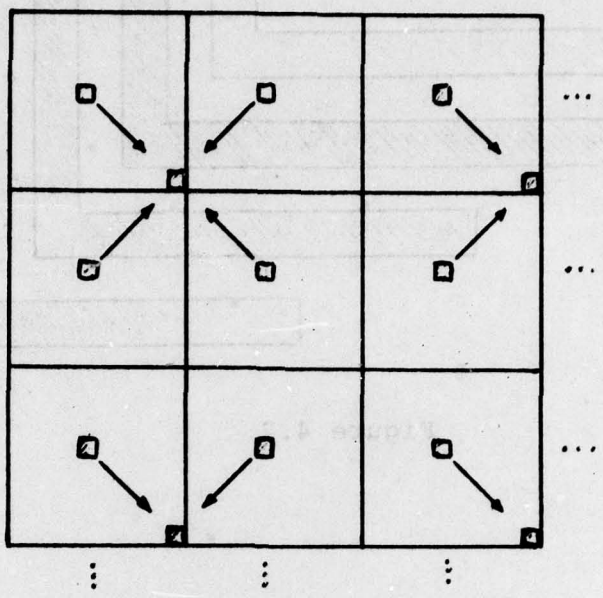


Figure 4.4

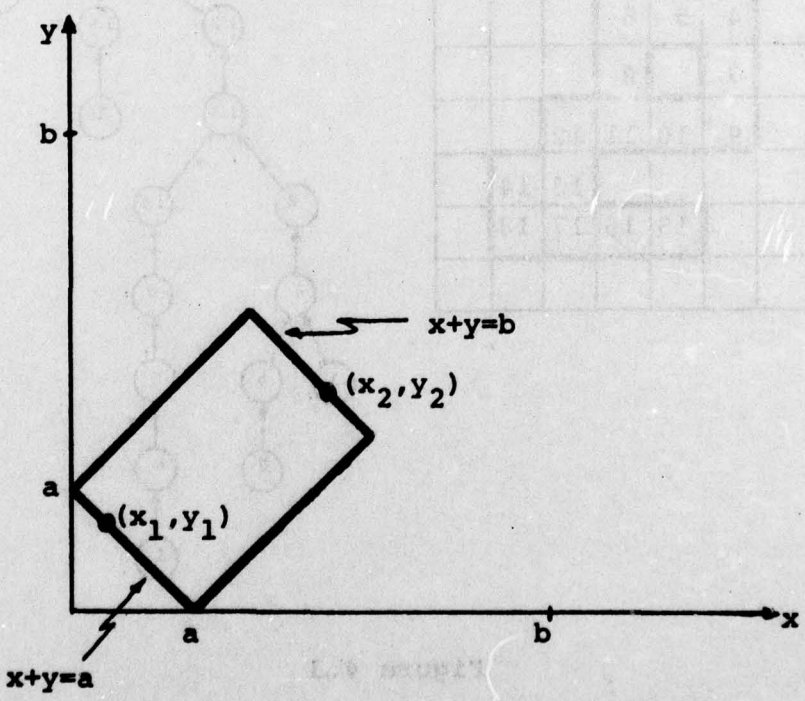


Figure 5.1

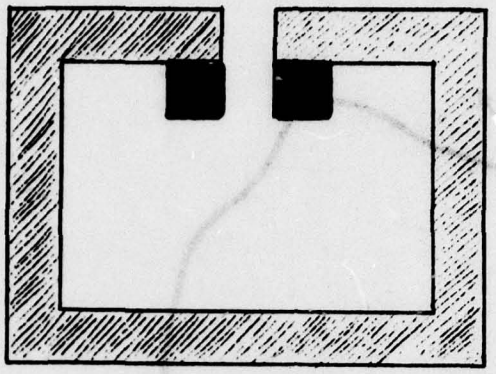


Figure 5.2

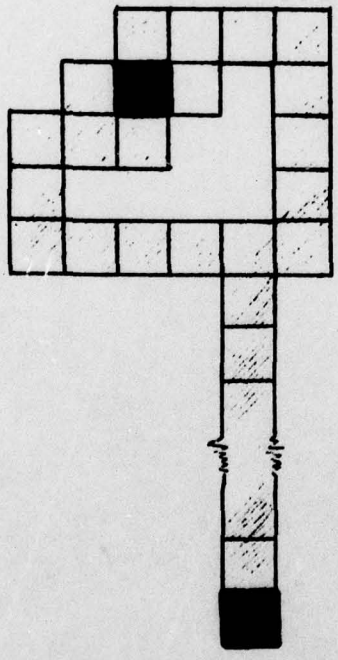


Figure 5.3

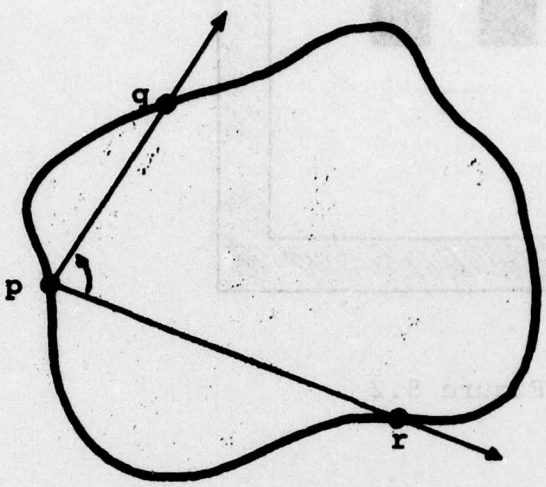


Figure 5.4

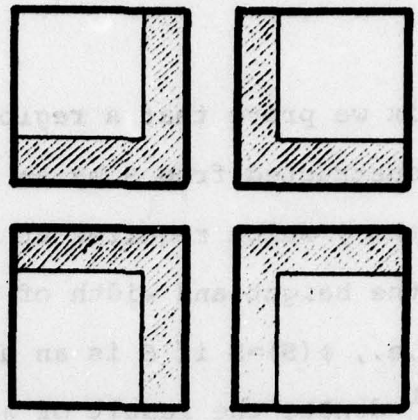


Figure 6.1

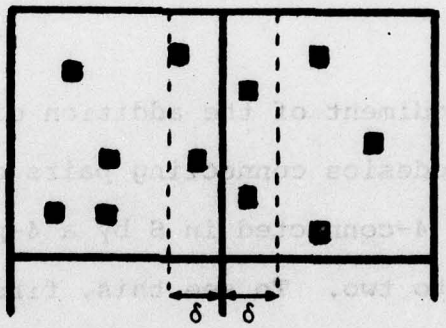


Figure 6.2

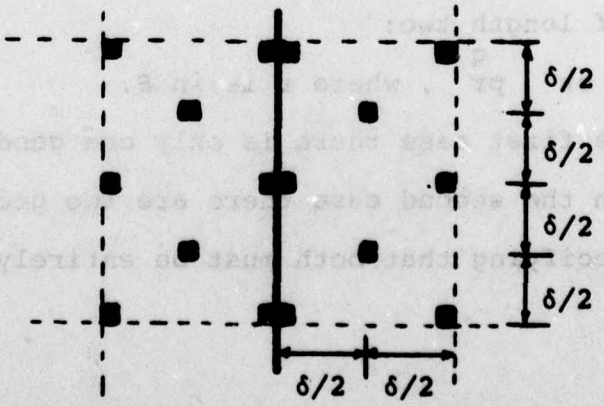


Figure 6.3

APPENDIX I

In this appendix we prove that a region S's upright framing rectangle can be constructed from S by an iterative parallel propagation algorithm ϕ which requires at most h+w-2 iterations, where h and w are the height and width of the rectangle, and which is stable, i.e., $\phi(S)=S$ if S is an upright rectangle. For brevity, let S^t denote the result of applying ϕ t times to S, i.e., $S^t=\phi^t(S)=\phi(\phi^{t-1}(S))$, where $\phi^0(S)=S$.

Define ϕ to be the parallel application of the rule (and all 90° rotations of it):

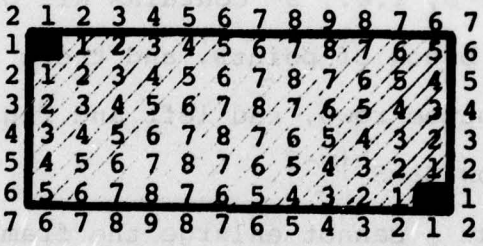
```
10   φ   11
11   →   11
```

This rule is an embodiment of the addition of all points in $\overline{S}$ which are in 4-geodesics connecting pairs of points p,q in S which are already 4-connected in S by a 4-path of length less than or equal to two. To see this, first notice that trivially there is a unique geodesic connecting p and q if they are at distance less than or equal to one. There are only two types (disregarding 90° rotational equivalents) of 4-paths from p to q of length two:

$$prq \quad \text{or} \quad \begin{matrix} q \\ pr \end{matrix}, \text{ where r is in S.}$$

Clearly in the first case there is only one geodesic joining p to q, and in the second case there are two geodesics, the above rule specifying that both must be entirely in S^1.

We now prove that S^t consists of all points in the picture which are in 4-geodesics connecting pairs of points in S which are 4-connected in S by a 4-path of length less than or equal to t+1. The proof is by induction on t. We have just proved the basis step when t=1. Now assume it holds for t=k-1. This means that for each pair of points p,q which are connected in S by a 4-path of length less than or equal to k, every 4-geodesic between p and q is entirely contained in S^{k-1}. The union of points on 4-geodesics between p and q is easily seen to be the upright rectangle with p and q in opposite corners, for example



Now consider a pair of points p,q whose shortest 4-path in S connecting them has length k+1, $p=p_0,p_1,\ldots,p_k=q$. Since there is a 4-path in S of length k connecting p_0 to p_{k-1}, by the induction hypothesis all of the points in the rectangle defined by these two diagonal corners are in S^{k-1}. These points are shown in Figure 1 with vertical hatching.

Similarly, since there is a 4-path in S of length k connecting p_1 to p_k, all of the points in the rectangle with diagonal corners at p_1 and p_k are already in S^{k-1}. These points are shown in Figure 1 with horizontal hatching. It is now easily verified, from the fact that p_1 is a 4-neighbor of p_0 and p_{k-1} is a 4-neighbor of p_k, that at most a single corner point of the rectangle of points defined by the pair of opposite corners (p,q) is not in S^{k-1}. By definition of ϕ it will be filled in during iteration k.

Define S* to be the union of S with those points in $\overline{S}$ which are on 4-geodesics connecting all pairs of points in S. To complete the proof we must show that S* is the upright framing rectangle of S, i.e., S* contains all of S, is an upright rectangular block of points, and there are points of S in S*'s top and bottom rows, and left and right columns. Finally, we must show $S^* = S^{h+w-2}$.

Notice first that ϕ cannot enlarge the framing rectangle of S. For example, consider a point p in $\overline{S}$ which is in the row above the top row of S's framing rectangle. Clearly this point can never become part of S* since this would require there to be a point adjacent to p in the same row which previously became part of S^k, for some k. Hence S* cannot include points outside of S's framing rectangle.

To see that S* contains all of the points in S's framing rectangle, consider four points of S, t, b, ℓ, r, that are in

the top and bottom rows, and left and right columns, respectively, of S's framing rectangle. (If there is more than one point on a side, choose one arbitrarily.) Since points t and ℓ are connected in S there exists a k such that the union of all points on 4-geodesics from t to ℓ is contained in S^k. Earlier we noticed that this is just the rectangle of points with opposite corners at t and ℓ. So this just fills in the top left corner of S's framing rectangle. Thus the union of the rectangles defined by the pairs (t,ℓ), (t,r), (b,ℓ), (b,r), and (ℓ,r) must include all of the points in S's framing rectangle, as shown in Figure 2. Thus S* contains just the points in S's framing rectangle.

Finally, Rosenfeld [25] proved that this procedure is completed after at most h+w-2 time steps, i.e., $S^*=S^{h+w-2}$. We briefly review that proof here for completeness. Let T=S*-S. First, observe that there cannot exist a pair of points p,q in T which are on opposite borders of S* with p 4-connected in T to q, since this would disconnect S. From this it follows that for each point $p \in T$ there is a quadrant (p is the origin of the coordinate system) in which S surrounds p, i.e., p cannot reach the border of S* in this quadrant without passing through S. In this quadrant, find the longest 4-path from p in T in which only two adjacent directions of movement are used. For example, in the northeast quadrant, the path must consist of upward and rightward moves only.

Since the path always remains in S*, its length is less than h+w-2. Furthermore, it terminates at a concave corner of S. Thus at the next iteration, this path is shortened. Repeating this argument, we see that p's longest path to a concave corner decreases by 1 at each iteration. In particular, after at most h+w-2 iterations, S's propagation has added p, i.e., p is contained in S^{h+w-2}.

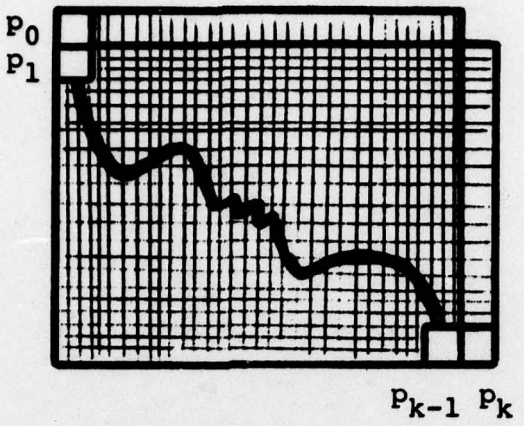


Figure 1

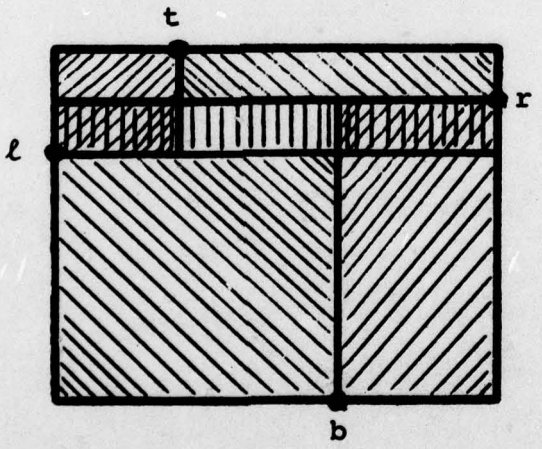


Figure 2